Université de Fribourg, Suisse
Département d'informatique
Diplôme complémentaire en informatique de gestion

Conception et implémentation d'un site de recrutement virtuel

Yvette Feldmann-Ossele
Rue Georges-Jordil 5
1700 Fribourg
yvette.feldmann@gmx.ch

Prof. Dr. Andreas Meier (1er rapporteur), à déterminer (2ème rapporteur)
Stefan Huesemann (assistant)

2 avril 2002

Tables des matières

Tables des figures

1 Introduction

1.1 Enoncé du problème

En général, les méthodes classiques de recrutement consistent à consulter les agences de recrutement locales ou les médias traditionnels. Le recrutement à travers ces méthodes est loin de satisfaire objectivement les besoins des employeurs et des employés. Les employés qualifiés n'étant pas toujours informés sur les offres disponibles sur le marché du travail. Il a été nécessaire de trouver une méthode rapide et efficace : d'où l'extension du recrutement à travers le net.

Les acteurs du marché des offres d'emploi ont compris les avantages compétitifs qu'Internet pouvait leur apporter. Un site d'emploi, au même titre qu'un journal, permet d'afficher des offres d'emploi. Mais en plus, il donne la possibilité de les renouveler en temps réel et d'en assurer le suivi. Le candidat peut consulter les offres en ligne et déposer immédiatement sa candidature en ligne. Ces raisons ont conduit les premiers acteurs du recrutement en ligne à créer des sites d'emploi pour regrouper le plus grand nombre d'offres possibles. Les premiers sites helvétiques souffleront leur 7ème bougie cette année. L'Internet est une amélioration des méthodes traditionnelles dans ce sens qu'il n'est pas géographiquement limité, son accès rapide et immédiat permet d'optimiser le processus de recrutement. La qualité des candidatures est maximisée grâce à la diffusion via l'Internet, un médium de haute technologie.

Le recrutement à travers les sites d'emploi en ligne permet d'offrir simultanément les avantages aux employeurs et aux employés.
Pour les entreprises, cette méthode garantit une visibilité optimale parmi les personnes correspondant au profil recherché par l'entreprise. Elle permet également une diffusion des annonces à moindre coût.
Pour les candidats, cette méthode les oriente facilement vers les emplois répondant à leurs aspirations et à leurs attentes.
Du fait que ces sites d'emplois regroupent sous un même toit des chercheurs et des pourvoyeurs d'emplois, ils constituent des outils de choix non seulement pour les entreprises à la recherche des candidats qualifiés, mais également pour les chercheurs d'emploi qui désirent être connus par ces employeurs.

1.2 Buts

Malgré le fait que les agences en ligne soient hautement performantes, il y subsiste des problèmes liés à la gestion des données. L'existence de ces problèmes nous amène à mettre en place un système de recrutement en ligne qui soit plus performant. Il serait bénéfique d'installer un site d'emplois, capable de proposer un recrutement rapide et interactif.

L'objectif de ce travail est de montrer à travers un exemple l'implémentation et l'utilisation d'un espace virtuel de recrutement.
Quels sont les acteurs du recrutement virtuel? Comment fonctionne un site de recrutement virtuel? Quels sont les aspects importants de la création d'un site d'emplois? Comment développer un catalogue d'emplois en ligne? Quels sont les apports du langage UML dans le développement d'une application WEB?

L'ambition de ce travail est de répondre à ces questions dans un contexte ou l'Internet modifie en profondeur le marché du recrutement.
Cette œuvre s'adresse aux étudiants en informatique ou en informatique de gestion ainsi qu'aux développeurs Web qui souhaitent connaître les possibilités offertes par le recrutement en ligne.

1.3 Plan

Le chapitre 2 – Les acteurs du recrutement en ligne. Ce chapitre est une description générale des ressources du marché d'emploi disponibles sur Internet. Les moteurs de recherche, les portails, les forums de discussion, les pages d'entreprises, les sites de média, les agences de placement de personnel et les marchés virtuels d'emploi sont présentés.

Le chapitre 3 – Analyse des fonctionnalités offertes par les marchés virtuels d'emploi. L'accent est mis sur les marchés virtuels d'emploi. Ce chapitre est une présentation des services offerts par les sites d'emploi en ligne.

Le chapitre 4 – Description des scénarios du système. Ce chapitre développe les possibilités offertes par un site d'emploi implémenté. Les fonctionnalités du système de recrutement en ligne sont décrites à partir des diagrammes de cas d'utilisation du langage de modélisation UML.

Le chapitre 5 – Techniques d'implémentation. Ce chapitre présente les techniques utilisées pour l'implémentation d'un marché virtuel d'emploi. Il s'agit d'une brève présentation des technologies suivantes : UML, serveurs, JSP, Struts et java.

Le chapitre 6 – Implémentation. Ce chapitre décrit la base de données et l'interface Web du marché virtuel d'emploi implémenté.

Le chapitre 7 – Mode d'emploi. Ce chapitre est un support utilisateur.

Le chapitre 8 – Conclusion. Ce chapitre est un résumé du travail effectué.

1.4 Notations

- Les expressions de langue anglaise sont écrites en *italique*.
- Le style `Courier New` est utilisé lorsque des extraits de code source ou des commandes sont mentionnés. Ce style est également utilisé pour les URL et les noms de fichier.
- L'énumération des figures se fait par chapitre. Il en va de même pour les tables.

2 Les acteurs du recrutement en ligne

On distingue plusieurs acteurs de l'emploi sur Internet: les moteurs de recherche, les portails, les forums de discussion, les pages d'entreprises, les sites de média, les agences de placement de personnel et les marchés virtuels d'emplois.
Ces acteurs sont analysés maintenant.

2.1 Les moteurs de recherche

D'après [Dhoquois 2000], il existe trois catégories de moteurs de recherche:

- Les annuaires de sites sont des listes d'adresses accompagnées parfois d'un petit commentaire. Ils sont réalisés par une personne ou une équipe chargée d'actualiser l'information. Leur utilisation s'apparente à une recherche dans un annuaire classique et ils proposent également des recherches thématiques. A moins que l'on ne recherche un site ou un secteur précis, cette démarche s'avère plus rationnelle que la recherche par moteur. L'indexation humaine, à la différence de l'indexation informatique, propose en effet un classement par échelle de valeur. C'est dans cette catégorie que se range Yahoo, Magellan, Voilà, Nomade (France) ou Enfin. Ce dernier a répertorié tous les annuaires, les méta-moteurs et les moteurs de recherche sur le Net francophone.
- Les moteurs de recherche disposent d'un robot qui recense selon une procédure automatisée le contenu internet. Ces moteurs de recherche permettent à l'utilisateur de sélectionner des mots ou des combinaisons de mots. De cette façon, il peut trouver des marchés virtuels d'emploi ou des informations sur les agences de placement de personnel, les entreprises et les individus. Les moteurs diffèrent quant aux possibilités d'entrer les données pour la recherche, dans leur manière de procéder, de visualiser les résultats et dans la rapidité. Plus la recherche est affinée, plus les résultats seront précis. Certains moteurs sont spécialisés dans une seule langue, comme Écila en France. D'autres sont accessibles dans plusieurs langues, comme Altavista, mais ne couvrent pas une zone géographique précise. Leur particularité, c'est qu'ils recensent des pages Web et non des sites. Le nombre de réponses données est donc plus important.
  Voici la liste de quelques moteurs de recherche :
  - `www.search.ch`
  - `www.google.ch`
  - `www.altavista.com`
  - `www.lycos.com`
  - `www.google.com`
  - `www.alltheweb.com`

- Les méta-moteurs de recherche permettent d'interroger plusieurs index à la fois. Ces services existent aussi bien pour des domaines spécifiques que pour les offres d'emploi. Des méta-moteurs recherchent des postes de travail en puisant simultanément dans différents marchés d'emploi sur Internet. L'utilisateur bénéficie d'un affichage sous un même format et ne doit interroger qu'une seule source. Les résultats sont en général bien structurés et le gain de temps est important. Néanmoins, les méta-moteurs ne consultent pas nécessairement toutes les bourses virtuelles d'emploi, certaines annonces pourraient donc échapper à l'intéressé.

Voici la liste de quelques méta-moteurs pour recherche d'emplois:

- `www.worldwidejobs.com`
- `www.metacrawler.com`
- `www.copernic.com`

2.2 Les portails

Les sites portails rassemblent souvent divers types d'informations dans le but d'attirer les internautes. Ils se veulent les portes d'entrée du Web vers des domaines plus ou moins étendus. La plupart des grands sites portails sont personnalisables : l'internaute choisit de n'afficher que les thèmes qui l'intéressent. Les sites portails créent aujourd'hui d'énormes bases de données d'offres d'emploi et de CV (curriculum vitae). C'est le cas de `Yahoo.com` sous la rubrique Yahoo Employement Arena.

2.3 Les forums

Un forum ou *newsgroup* est un lieu de discussion et d'échange entre les internautes. Ces forums portent en général sur des thèmes précis. Ces forums de discussions sont regroupés dans un gigantesque réservoir d'articles et de contributions écrites : le *Usenet*, un sous-ensemble d'Internet.
Il existe deux possibilités d'accès à un forum de discussion : l'accès par le logiciel de navigation ou un logiciel de lecture des *newsgroup* (ce logiciel est intégré dans la plupart des *browsers*) et l'accès par le Web.
En ce qui concerne l'accès par le *browser* (navigateur), l'utilisateur doit choisir un serveur de *newsgroups*. Le serveur contient la liste des *newsgroups* accessibles.
Quant à l'accès par le Web, il suffit de se connecter dans un moteur de recherche des *newsgroups*. Il existe des *newsgroups* spécialisés pour la recherche d'emploi. C'est le cas de la rubrique `misc.jobs` sur `[Google 2001]`. Tout internaute peut poster ses requêtes qui sont par la suite ventilées sur le réseau.

2.4 Les pages d' entreprises

Aujourd'hui, les grandes entreprises utilisent Internet pour recruter. Les employeurs potentiels proposent sur leur site Web des offres d'emploi intéressantes et un nombre considérable d'informations : historique de l'entreprise, chiffres clés (effectif, chiffre d'affaires, etc.), organigramme, descriptif des différentes entités et des différents métiers qui les composent, valeurs de l'entreprise, etc..

2.5 Les sites de média

La majorité des hebdomadaires et des journaux ont des sites Internet où ils affichent leurs annonces. Les deux sites suivants regroupent la majorité des journaux suisses:
`www.jobclick.ch`, `www.zannonces.ch`.
Les presses locales possèdent également leur propre site. Dans la région de Fribourg, on trouve `www.laliberte.ch`.

2.6 Les agences de placement de personnel

Les agences de placement de personnel sont les agences d'intérim et les cabinets de recrutement. Les entreprises s'adressent à ces prestataires de services lorsqu'elles n'ont pas de service spécialisé en ressources humaines. Leur mission est donc de sélectionner un ou plusieurs candidats pour le compte d'une entreprise.

2.7 Les marchés virtuels d'emploi

L'expansion de l'Internet sur le marché de l'emploi a donné naissance aux sites de recrutement en ligne. Ce sont des intermédiaires entre l'offre et la demande sur le marché de l'emploi. Ce nouveau mode de recrutement a révolutionné le marché du travail.
D'après [Recto-Verso2001], Il existe en Suisse plus de 60 bourses d'emploi. Certains marchés virtuels d'emploi sont spécialisés par secteur, d'autres plus généralistes proposent plusieurs secteurs d'activités.

2.7.1 Identification des sites généralistes

En Suisse, les principaux sites généralistes sont :
`www.jobpilot.ch`;
`www.monster.ch`;
`www.idealjob.ch`;
`www.jobscout24.ch`;
`www.stepstone.ch`;
`www.topjobs.ch`.

Ayant tous une présence internationale, ils offrent également l'accès à des sites pour d'autres pays, permettant de rechercher un poste à l'étranger.

2.7.2 Identification des sites spécialistes

Les sites spécialistes sont des marchés d'emploi destinés au jeunes diplômés ou restreints à des secteurs particuliers.
L'adresse `www.yoodle.ch` permet de consulter une liste des sites spécialisés.

Voici quelques exemples de marchés d'emplois spécialisés dans certains secteurs :
`www.lawjobs.ch` (Droit) ;
`www.mcjob.ch` (Banking/Consulting) ;
`www.nexus.ch` (IT) ;
`www.persoenlich.com` (Marketing/Publicité).

La liste qui suit présente les marchés d'emplois orientés vers les jeunes diplômés :
`www.diplom.ch` ;
`www.jobwinner.ch` ;
`www.vip.unisg.ch` ;
`www.swiss-science.org` ;
`www.skillworld.com` ;
`www.success-and-career.ch` ;
`www.talents.ch` ;
`www.telejob.ch`.

Nous délimitons le cadre de ce travail aux sites d’emploi en ligne. C’est la raison pour laquelle une analyse plus détaillée de l’ensemble de ces sites nous semble opportune.

3 Analyse des fonctionnalités offertes par les marchés virtuels d'emploi

Les critères principaux à prendre en compte pour analyser les fonctionnalités des différents sites d'emploi en ligne sont : la présentation, les fonctionnalités, la rapidité et l'efficacité du traitement d'offres et demandes.

3.1 Présentation

Q'ils soient généralistes ou spécialisés, les sites d'emploi en ligne ont, en règle générale, la même présentation. L'utilisateur y trouvera une rubrique consacrée aux candidats et une autre consacrée aux entreprises.
La rubrique consacrée aux candidats leur permet de déposer leurs CV en ligne et de consulter les offres d'emploi disponibles.
La rubrique consacrée aux entreprises leur permet de déposer les offres d'emploi et de consulter les CV.
La figure 3.1 résume les principales rubriques d'un site de recrutement en ligne.

Figure 3.1: la page d'accueil du site de recrutement `idealjob.ch`.

En général, les sites sont bien présentés. La plupart présentent une cohérence graphique et une navigation intuitive.

3.2 Les fonctionnalités

Le service offert est différent selon que l'utilisateur est employeur ou chercheur d'emploi.

3.2.1 La Cvthèque

Les Cvthèques permettent aux recruteurs d'élargir le champ de leur prospection. L'accès à cette base de données est gratuit ou payant selon les cas. La diffusion des CV pose un problème de confidentialité qui peut être en partie résolu par l'utilisation des masques de saisie. Ce qui permet de cacher les coordonnées du candidat.

3.2.2 Les moteurs de recherche

La principale fonctionnalité d'un site d'emploi pour les candidats est la recherche d'offres. Les sites regroupent ces offres dans des bases de données consultables par mots-clés. Le moteur identifie dans la base toutes les offres incluant les mots-clés demandés par le candidat.
La même technique est utilisée par le recruteur lorsqu'il consulte les Cvthèques : il propose une liste de mots-clés et le moteur fait remonter tous les CV conformes à sa recherche.
Les moteurs sont aussi appelés "agents intelligents" ou "robots intelligents". Ce sont en réalité des programmes informatiques qui comparent des champs ou des caractères. De manière générale, il existe deux catégories de moteurs de recherche.

La première catégorie prend peu de critères en compte. Ce genre de moteur filtre peu, récupérant dans la base de données tout ce qui touche de près ou de loin à la demande. Il en résulte ainsi une liste contenant les postes ne correspondant pas du tout au profil souhaité. Par conséquent, le candidat devra procéder à tri fastidieux.
Ce type de moteur dans un site d'emploi est intéressant pour les jeunes diplômés, qui peuvent s'ouvrir à plusieurs carrières. Cependant, les candidats peuvent aussi se lasser et laisser passer une offre intéressante parmi des dizaines d'offres hors profil. C'est le cas du site d'emploi [EmailJob 2002].

La deuxième catégorie de moteurs fonctionne avec des critères de sélection beaucoup plus précis : secteur d'activité, fonction, salaire, localisation, etc.. Ils traquent les offres correspondant exactement au profil défini.
Ce type de recherche convient aux expérimentés et aux cadres, qui ont des projets professionnels bien définis et peu de temps pour trier les offres inutiles. Mais le risque d'une recherche trop précise est de rentrer bredouille et de vouloir changer de site. C'est le cas du site d'emploi [Cadre Emploi 2002].
La solution réside dans l'utilisation du moteur. Plus il est sélectif, mieux on peut le maîtriser pour affiner ou élargir la recherche.
Si le moteur n'est pas sélectif, on n'a pas de moyen d'affiner la recherche.

3.2.3 Le push-mail

C'est la possibilité de recevoir des offres sur sa messagerie. Pour être efficace, le *push-mail* doit être rapide : les offres doivent parvenir au candidat dans la journée de leur parution en ligne. Sur certains sites, l'opération est simultanée.
Le *push-mail* doit également être précis : une qualité qui dépend des robots intelligents et des informations demandées sur le CV électronique.
De la même manière, les recruteurs peuvent recevoir de nouveaux CV ciblés, par *e-mail*, en réponse à leurs offres.

3.2.4 Le dépôt d'offres en ligne

Tous les sites ne proposent pas de service de dépôt d'offres en ligne. Un appel téléphonique ou un courrier restent parfois nécessaires pour certains sites. L'annonce reste en ligne pour

une durée déterminée et le service est souvent payant. Ces offres décrivent le profil du candidat recherché.

3.2.5 La mailing-list

Des logiciels de présélection permettent d'identifier rapidement les bons CV et, à partir de cette sélection, d'établir une *mailing-list*. Les recruteurs peuvent ainsi envoyer électroniquement une annonce à un panel choisi de candidats. Certains de ces logiciels apprécient les compétences des candidats grâce à un formulaire d'évaluation en ligne.

Certains sites offrent souvent des services additionnels : présentation de quelques entreprises, l'emploi à l'international, actualité sur l'emploi, conseils sur les CV et les lettres, participation aux forums de discussion.

3.3 Rapidité et efficacité du traitement d'offres et demandes

Les services offerts au candidat par l'agence ne sont malheureusement pas toujours rapides. En effet, après avoir trouvé une offre d'emploi, le candidat ayant postulé au travail souhaité ne reçoit pas toujours une réponse immédiate. Très souvent, l'offre affichée n'est plus disponible faute d'une mise à jour régulière.
Les fonctions d'alerte qui donnent la possibilité au candidat de définir un profil de recherche s'avèrent parfois peu efficace.
Il faut noter que les services offerts aux candidats sont entièrement gratuits.
L'entreprise quant à elle reçoit un traitement de faveur. L'agence sélectionne les candidats les plus performants, en prenant soin de respecter les délais exigés par le recruteur.

3.4 L'architecture

Dans un site d'emploi en ligne la recherche peut être faite de manière centralisée ou distribuée.

3.4.1 Architecture d'une recherche centralisée

Il s'agit d'une recherche faite dans une base de données unique. L'utilisateur propose une liste de mots-clés au moteur de recherche et le moteur fait remonter les résultats conformes à sa recherche. Presque tous les marchés d'emploi, généralistes ou spécialistes sont implémentés suivant ce modèle.
La figure 3.2 nous montre le schéma de l'architecture élémentaire d'une recherche centralisée.

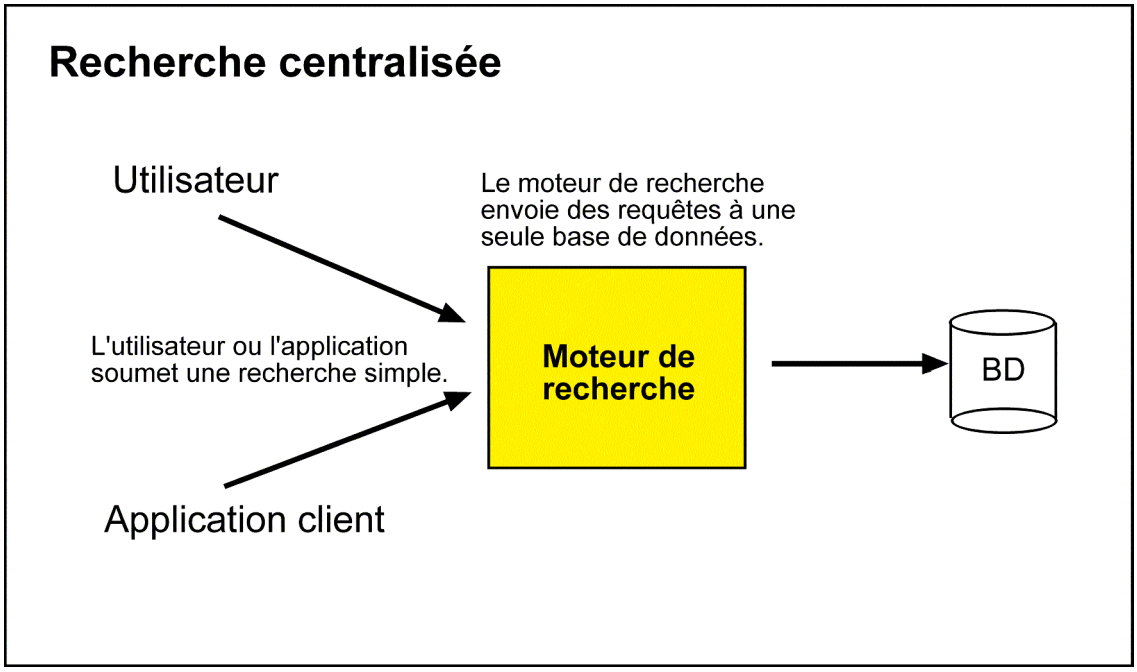


Figure 3.2: architecture d'une recherche centralisée.

3.4.2 Architecture d'une recherche distribuée

Il s'agit d'une recherche faite de façon simultanée dans plusieurs bases de données.
La requête pour une telle recherche sera envoyée à plusieurs sites en même temps. Lorsque leurs réponses parviendront au méta-moteur, ce dernier fusionnera, formatera et présentera les résultats.
Le schéma de l'architecture élémentaire de la recherche distribuée de la figure 3.3, nous montre un client qui, par le biais d'un méta-moteur, explore trois bases de données situées à des endroits différents.

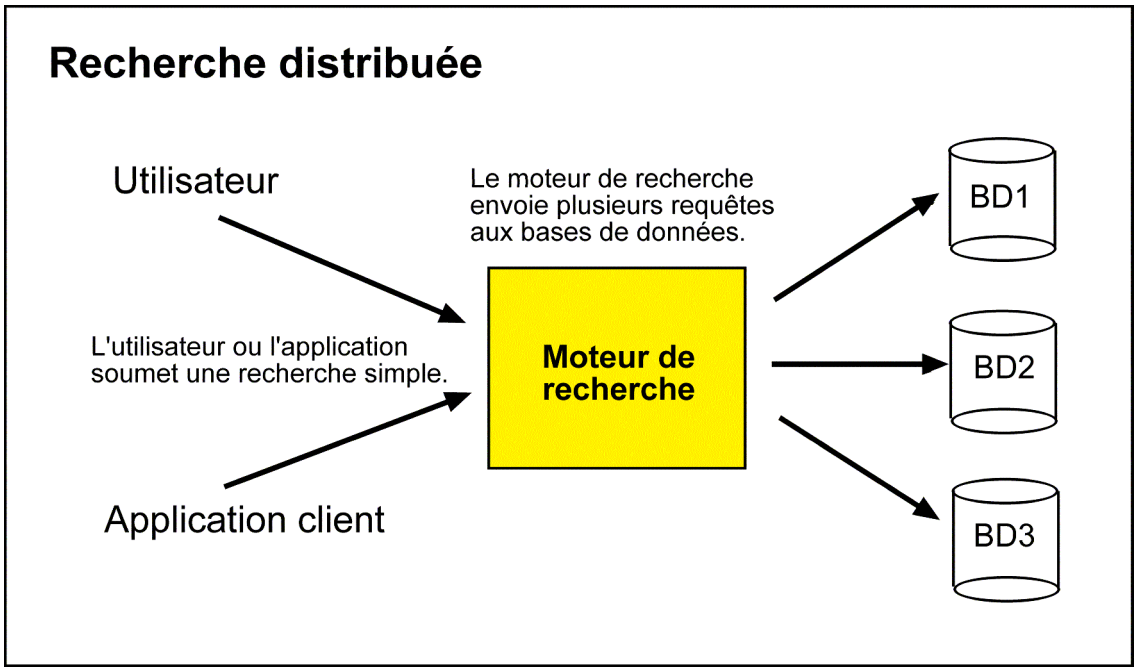


Figure 3.3: architecture d'une recherche distribuée.

La liste qui suit présente les bourses d'emplois avec chercheurs multi-sites :

- `www.emploi.ch`. La page d'accueil fournit directement la rubrique Emploi-Crawler.

- `www.netjobs.ch`. Le moteur de recherche en langue allemande assure la recherche sur 20 sites d'emploi en Suisse. Trois clés de recherche sont disponibles, dont une par mot-clé.
- `www.search11.com`. Ce site contient de très nombreuses offres et permet une recherche simultanée sur plusieurs grandes bourses d'emploi suisses. Le moteur de recherche permet de définir plusieurs critères.

En résumé, les sites d'emploi en ligne représentent un moyen révolutionnaire pour le marché de l'emploi. Néanmoins, certains points devraient être améliorés pour gérer toutes les données générées sur l'Internet.

4 Description des scénarios du système

L'agence à réaliser dans le cadre de ce travail va intégrer les solutions diverses existant sur le marché afin d'optimiser l'ensemble des processus *online* (en ligne).

La dénomination donnée est jobplace. Ce site de recrutement *online* s'est fixé les objectifs suivants: mettre en place un espace d'échanges interactif afin de faciliter le processus de recrutement, accompagner les entreprises pour optimiser leurs recrutements et donner aux candidats des moyens efficaces et conviviaux pour gérer leur carrière.

Pour réaliser ces objectifs, jobplace va offrir aux candidats et aux recruteurs plusieurs fonctionnalités qui sont résumées dans le tableau 4.1.

Acteurs	Services
Candidat	Dépôt de CV en ligne Envoi d'offres d'emploi Conseils CV/Lettre/Entretien Personnalisation des informations Informations marché de l'emploi
Recruteur	Dépôt d'offres d'emploi en ligne Consultation de Cvthèque Envoi de CV Statistiques en ligne Personnalisation des informations Page de présentation des sociétés

Tableau 4.1: les fonctionnalités offertes par jobplace.

Nous présenterons dans un premier temps le service proposé aux candidats et dans un second temps le service proposé aux recruteurs.

Tout candidat, inscrit ou non a la possibilité de consulter les offres d'emploi.
Les candidats inscrits reçoivent un compte personnel de gestion des données. Ils peuvent aussi consulter les offres d'emploi ou recevoir directement dans leur compte celles qui correspondent à leur profil. Ils ont également la possibilité faire des postulations en ligne en déposant un CV et une lettre de motivation. Les critères saisis par le candidat se déversent directement dans une base de données consultable par code d'accès. Les dossiers de candidature sont envoyés au recruteur.
Dans le cas où le recruteur serait intéressé, il prend contact avec le candidat pour l'entretien.

Quant aux entreprises, elles ont la possibilité de s'inscrire pour recevoir un compte et des candidatures, déposer les annonces et visualiser les CV publiés.
L'employeur choisit par la suite les candidats pour une entrevue.

4.1 Les fonctionnalités du système : les diagrammes de cas d'utilisation

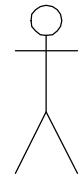
Les fonctionnalités du système sont décrites à partir de diagrammes de cas d 'utilisation ou *use case diagrams* du langage de modélisation objet UML (Unified Modeling Language). Les

diagrammes de cas d'utilisation illustrent les fonctions du système *(*cas d'utilisation), leurs limites (acteurs) et les relations entre les cas d'utilisation et les acteurs.

La modélisation du système commence par l'identification des acteurs et des cas d'utilisation et se poursuit par la description des cas d'utilisation.
Pour une bonne compréhension du modèle, il paraît nécessaire de définir certains termes propres au langage UML.

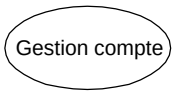
4.1.1 Les concepts fournis par les cas d'utilisation

1. Les Acteurs. Ils n'appartiennent pas au système, mais ils interagissent avec celui-ci. Ils fournissent de l'information en entrée et/ou reçoivent de l'information en sortie.
Dans UML, l'acteur est représenté comme suit :



Le nom de l'acteur correspond au rôle joué par la personne.
Les différents acteurs retenus dans le système de recrutement *online* sont : le candidat, le recruteur, l'employé d'agence et l'administrateur.

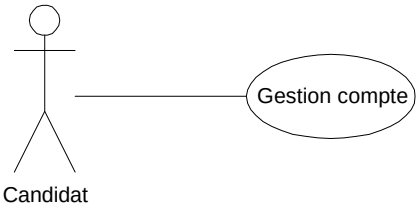
2. Les cas d'utilisation ou *use cases*. Un cas d'utilisation modélise un dialogue entre un acteur et le système. C'est la représentation d'une fonctionnalité offerte par le système. L'ensemble des cas d'utilisation forme toutes les façons possibles d'utilisation du système.
Dans UML, le cas d'utilisation est représenté par un ovale comme suit:



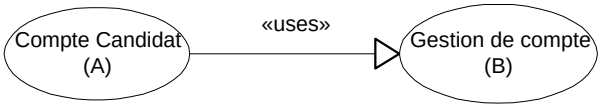
Les cas d'utilisation identifiés dans le système de recrutement en ligne sont décrits dans le diagramme de cas d'utilisation (voir 4.1.2).

3. Les relations dans les cas d'utilisation. UML propose différents types de liens entre les acteurs et les cas d'utilisation : la relation de communication, la relation d'utilisation et la relation d'extension.

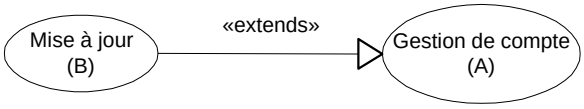
La relation de communication indique la participation d'un acteur et est représentée par une ligne solide entre l'acteur et le cas d'utilisation. C'est la seule relation possible entre un acteur et les cas d'utilisation.



La relation d'utilisation ou *uses* entre cas d'utilisation signifie qu'une instance du cas d'utilisation source inclut aussi le comportement décrit dans le cas d'utilisation destination. Le cas d'utilisation A utilise (*uses*) le cas d'utilisation B signifie qu'une instance de A va engendrer une instance de B et l'exécuter. A connaît B, par ailleurs B ne connaît pas A, c'est-à-dire que A dépend de B.



La relation d'extension ou *extends* entre deux cas d'utilisation signifie que le cas d'utilisation source étend le comportement du cas d'utilisation destination.
Le cas d'utilisation B *extends* le cas d'utilisation A signifie que le comportement d'une instance de A peut être complété par le comportement d'une instance de B. Il s'agit de montrer un comportement optionnel qui se déroule sous certaines conditions. Une instance de A va engendrer une instance de B et l'exécuter sous certaines conditions. B connaît A et non l'inverse, c'est-à-dire que B dépend de A. B n'existe pas tout seul et A existe sans B.



4.1.2 Diagramme de cas d'utilisation

La figure 4.1 présente le diagramme des cas d'utilisation du système de recrutement en ligne.

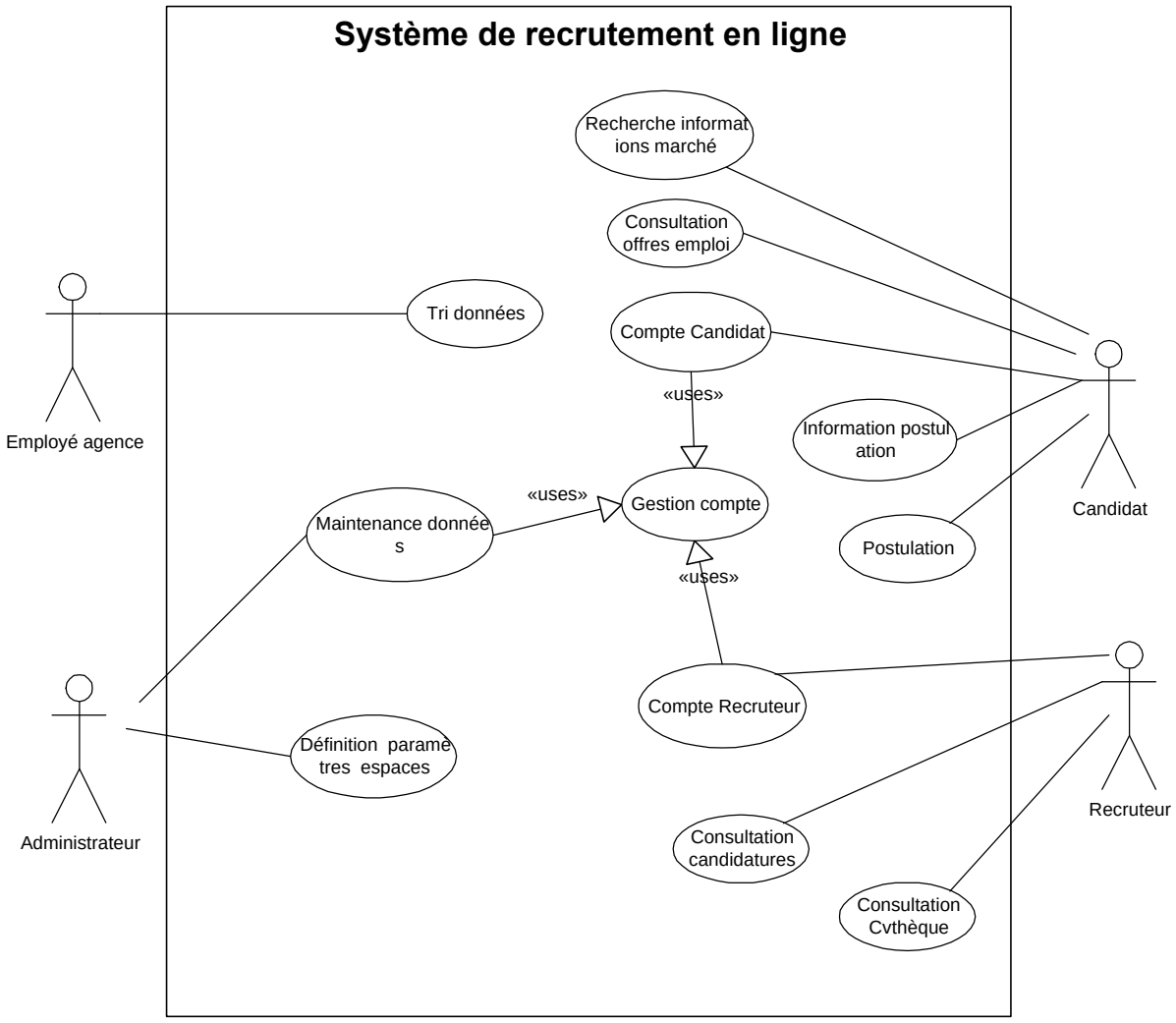


Figure 4.1: diagrammes des cas d'utilisation du système de recrutement en ligne.

4.2 La description des cas d'utilisation

4.2.1 Cas d'utilisation : Recherche informations marché

Il est activé par le candidat. Le site propose ce service grâce à un lien qui conduit directement sur la page Web du secrétariat d'Etat à l'économie [seco 2001].

4.2.2 Cas d'utilisation : information postulation

Il est activé par l'acteur Candidat. C'est le suivi de l'état des candidatures. Ce suivi synthétise toutes les offres JobPlace auxquelles le candidat a répondu et renseigne sur la suite donnée à ces candidatures (transmises à l'entreprise ou non, réponse négative de l'entreprise, convocation pour entretien etc.).

4.2.3 Cas d'utilisation : Gestion compte

Il est activé par le candidat ou le recruteur. Compte tenu de la complexité de ce cas d'utilisation, il paraît nécessaire de montrer les interactions avec les autres cas d'utilisation du système, d'où le diagramme des cas d'utilisation dans la figure 4.2.

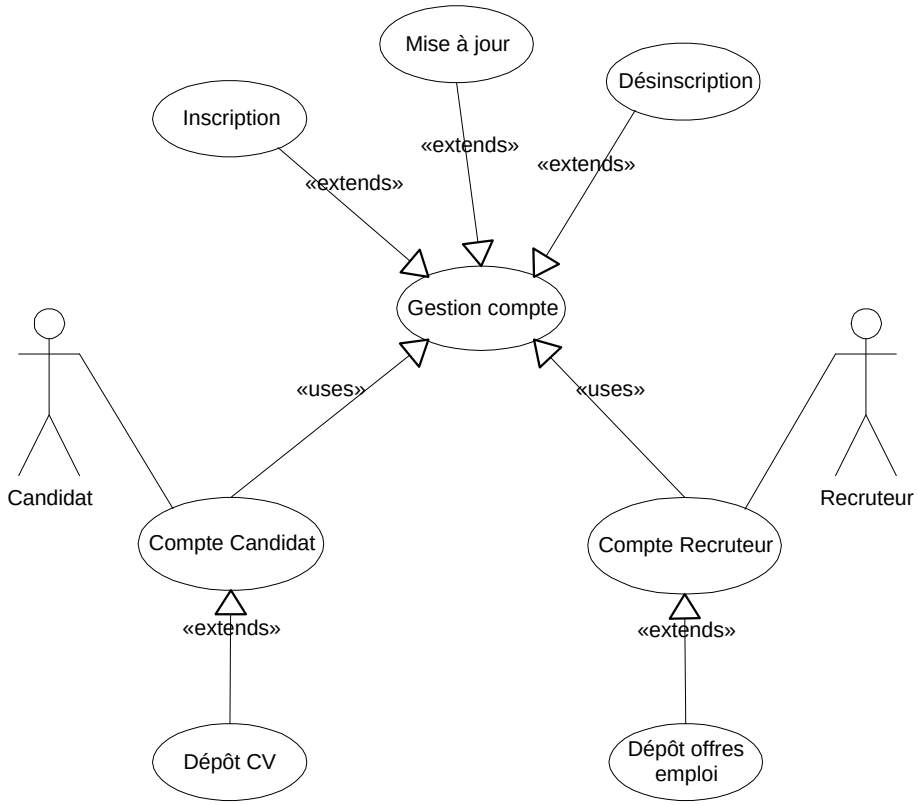


Figure 4.2: diagramme de cas d'utilisation gestion de compte.

La relation d'extension dans ce diagramme décrit un comportement optionnel. L'inscription est la pré-condition à tous les autres cas d'utilisation. Par la suite le candidat et/ou le recruteur peuvent procéder à une mise à jour des données du compte ou à la désinscription. Le candidat peut un CV dans son compte et le recruteur une offre ou plusieurs offres d'emploi.

4.2.4 Cas d'utilisation : Inscription

Il est activé par le candidat ou par le recruteur. Il s'agit de remplir un formulaire pour créer un profil utilisateur. Les informations classiques sont demandées : titre, nom, prénom, e-mail, nom d'utilisateur, mot de passe.

4.2.5 Cas d'utilisation : Mise à jour

Il est activé par le candidat ou par le recruteur. Il s'agit des modifications effectuées sur le profil utilisateur ou sur les données du compte.

4.2.6 Cas d'utilisation : Désinscription

Il est activé par le candidat ou par le recruteur. Il s'agit de la destruction du profil utilisateur.

4.2.7 Cas d'utilisation : Compte Candidat

Il s'agit de l'espace réservé au candidat. Chaque candidat enregistré peut accéder à un ensemble de services personnalisés. Seul le titulaire du mot de passe et du nom d'utilisateur correspondant à un compte candidat pourra accéder à cet espace. Les services suivants sont opérationnels :

- dépôt de CV ou de lettres de candidature (voir use case Dépôt CV)
- visualisation de CV ou de lettre de candidature
- modification de CV ou de lettres de candidature
- suppression de CV ou de lettres de candidature
- visualisation des offres d'emploi

4.2.8 Cas d'utilisation : Compte Recruteur

Il s'agit de l'espace réservé au recruteur. Le fonctionnement est le même que celui du compte Candidat sauf pour les services offerts. Tout recruteur enregistré peut accéder aux services suivants :

- visualiser les CV
- visualiser les candidatures
- classer et gérer les CV choisis
- déposer les offres d'emploi (voir *use case* Dépôt offres d'emploi)

4.2.9 Cas d'utilisation : Consultation offres emploi

Il est activé par le candidat. Pour consulter les offres d'emploi, rien de plus simple, pas besoin de s'enregistrer au préalable. Un vaste choix est proposé dans plusieurs départements. Le candidat dispose d'un sélecteur dont les critères sont :

- Le secteur d'activité. Un ascenseur permet à l'utilisateur de choisir celui qui l'intéresse.
- La région. Un ascenseur (menu déroulant) permet également à l'utilisateur de choisir la région qui l'intéresse.
- Une recherche par mots-clés. Il s'agit du nom de l'annonceur, du poste, de la région et de la date de publication de l'annonce.

Après avoir précisé ses critères, le candidat peut lancer la recherche à l'aide du bouton prévu à cet effet, et les annonces s'affichent ainsi sur l'écran. Les annonces sélectionnées peuvent être collectées dans un caddie (voir *use case* Caddie).

Les annonces sont en ligne pour une durée déterminée et sont régulièrement mises à jour. Les offres sont classées par ordre chronologique de publication, de la plus récente à la plus ancienne. Elles sont présentées d'une manière structurée.

Pour consulter les annonces dans le détail, il suffit de cliquer sur le titre de l'annonce pour visualiser la totalité.

4.2.10 Cas d'utilisation : dépôt offres emploi

Il est activé par le recruteur. Ce dernier doit remplir un formulaire pour créer une offre d'emploi. Ce formulaire contient les informations suivantes :

- Le titre de l'offre.
- La référence de l'offre de l'entreprise.
- Le type de poste. Un ascenseur permet de choisir le poste désiré.
- La description du poste
- La région. Un ascenseur permet au recruteur de choisir le lieu.
- Le type de contrat.
- L'horaire. Un ascenseur permet au recruteur de sélectionner l'horaire du poste.
- Le salaire minimum proposé.
- Le salaire maximum proposé.
- La date d'embauche.
- Le niveau d'études requis. Un ascenseur permet de sélectionner le niveau désiré.
- Les compétences demandées.
- Le secteur d'activité de la société recruteuse. Un ascenseur permet de sélectionner ce dernier.
- La taille de l'entreprise
- Le type d'entreprise
- Le nom de l'entreprise
- L'adresse Internet
- La description de la société
- Les coordonnées du contact.

4.2.11 Cas d'utilisation : Consultation candidatures

Il est activé par le recruteur. Ce dernier reçoit dans son compte les dossiers de candidature relatifs à une annonce en ligne.

4.2.12 Cas d'utilisation : Consultation Cvthèque

Il est activé par le recruteur. Pour consulter les CV, le recruteur doit s'inscrire au préalable. Le recruteur dispose d'un sélecteur dont les critères sont :

- Le poste. Un ascenseur permet au recruteur de choisir celui qui l'intéresse.
- La région. Un ascenseur permet également au recruteur de choisir la région qui l'intéresse.
- Le secteur d'activité. Un ascenseur permet également au recruteur de choisir la région qui l'intéresse.
- Une recherche par mots-clés.

Après avoir précisé ses critères, le recruteur peut lancer la recherche à l'aide du bouton prévu à cet effet, et les CV s'affichent ainsi sur l'écran. Les CV sélectionnés peuvent être collectés dans un caddie (voir *use case* caddie).

Les CV sont en ligne pour une durée déterminée, à compter de leur publication. Pour consulter les CV dans le détail, il suffit de cliquer sur le titre du CV pour visualiser la totalité. Les CV sont présentés de manière structurée.

4.2.13 Cas d'utilisation : Postulation

Il est activé par le candidat. Il faut au préalable être inscrit. Le candidat a la possibilité de postuler directement après consultation de la base de données des offres d'emploi. Il peut aussi postuler aux annonces qui se trouvent dans son compte. Il peut enfin postuler à partir du caddie
Pour la postulation proprement dite, le candidat doit déposer un CV et ajouter une lettre de motivation.
Il paraît nécessaire de construire un diagramme de cas d'utilisation (cf. figure 4.3) supplémentaire pour expliquer le fonctionnement du système.

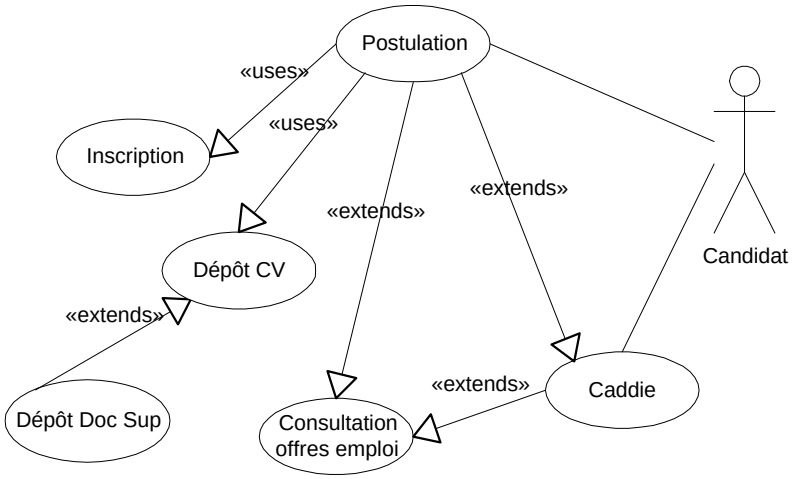


Figure 4.3: fonctionnement du système postulation.

4.2.14 Cas d'utilisation : Dépôt CV

Il est activé par le candidat. Il a le choix entre, consulter les annonces puis créer le CV et l'attacher aux annonces sélectionnées ou bien l'inverse, en créant au préalable le CV, puis en sélectionnant des offres. Le candidat peut déposer son CV sur une ou plusieurs offres ou encore le laisser sur la base, dans l'attente d'offres correspondant aux critères saisis, auquel cas il sera envoyé sur l'espace entreprise de l'annonceur concerné. Pour déposer un CV, le candidat doit remplir un formulaire. Les informations suivantes sont demandées dans le formulaire CV : état civil, formation, expérience professionnelle, motivation professionnelle, souhaits d'évolution, disponibilité et mot de passe.

4.2.15 Cas d'utilisation : Dépôt Doc Sup

Il est activé par le candidat. Après la consultation des offres d'emploi, le candidat peut créer la lettre de motivation et la relier à l'offre.

4.2.16 Cas d'utilisation : Caddie

Il est activé par le candidat ou par le recruteur. Il s'agit d'un panier virtuel servant à placer des offres d'emploi sélectionnées pour le candidat et les CV pour le recruteur. A tout moment l'utilisateur peut appuyer sur un bouton afin de voir le contenu du caddie.

4.2.17 Cas d'utilisation : Tri données

Activé par l'employé d'agence. Il doit trier les annonces afin d'envoyer au candidat les offres d'emploi correspondant aux critères mentionnés dans le CV.
Il doit également envoyer au recruteur les CV qui remplissent les critères de l'annonce.

4.2.18 Cas d'utilisation : Définition paramètres espaces

Il est activé par l'administrateur. Il doit actualiser les paramètres des espaces candidat et recruteur.

4.2.19 Cas d'utilisation : Maintenance données

Il est activé par l'administrateur. Ce dernier s'occupe de la mise à jour des offres d'emploi. Il peut ainsi afficher la date de la dernière mise à jour, mettre en valeur les nouveautés et afficher un compteur du nombre de visiteurs ou du nombre d'offres d'emploi.

5 Techniques d’implémentation

5.1 UML

UML (*Unified Modeling Language*), que l’on peut traduire par "langage de modélisation unifié" est une notation permettant la modélisation d’un problème. Ce langage est né de la fusion de plusieurs méthodes existant auparavant, et est devenu la référence en terme de modélisation objet.
Entre 1970 et 1990, de nombreux analystes ont mis au point des approches orientées objets, si bien qu'en 1994 il existait plus de 50 méthodes objet. Toutefois seules 3 méthodes ont véritablement émergées:

- La méthode OMT de Rumbaugh
- La méthode BOOCH'93 de Booch
- La méthode OOSE de Jacobson (*Object Oriented Software Engineering*)

A partir de 1994, Rumbaugh et Booch (rejoints en 1995 par Jacobson) ont uni leurs efforts pour mettre au point la méthode unifiée incorporant les avantages de chacune des méthodes précédentes.
La méthode unifiée à partir de la version 1.0 devient UML (*Unified Modeling Language*), une notation universelle pour la modélisation objet.
UML 1.0 est soumise à l'OMG (*Object Management Group*) en janvier 1997, mais elle ne sera acceptée qu'en novembre 1997 dans sa version 1.1, date à partir de laquelle UML devient un standard international.
Cette méthode représente un moyen de spécifier, représenter et construire les composantes d’un système informatique.
En effet, la notation unifiée définit 9 diagrammes pour représenter les différents points de vue de modélisation. Ces diagrammes permettent de visualiser et de manipuler les éléments de modélisation. Les diagrammes définis par UML sont les suivants :

- Les diagrammes de cas d’utilisation: représentation des fonctions du système du point de vue de l’utilisateur.
- Les diagrammes de séquence: représentation temporelle des objets et de leurs interactions.
- Les diagrammes d’activités: représentation du comportement d’une opération en terme d’actions.
- Les diagrammes de composants: représentation du code en termes de modules, de composants et surtout des concepts du langage ou de l’environnement d’implémentation.
- Les diagrammes de classes: représentation de la structure statique en terme de classes et de relations.
- Les diagrammes de collaboration: représentation spatiale des objets, des liens et des interactions.
- Les diagrammes de déploiement: représentation du déploiement des composants sur les dispositifs matériels.
- Les diagrammes d’états-transitions: représentation du comportement d’une classe en terme d’état.
- Les diagrammes d’objets: représentation des objets et de leurs relations, correspond à un diagramme de collaboration simplifié, sans représentation des envois de messages.

De façon plus générale, UML permet de modéliser:

- Les utilisations de cas et les scénarios (spécifications, architecture fonctionnelle);
- les classes et les objets (analyse technique détaillée);
- les composants (architecture logicielle);
- la distribution et le déploiement (architecture technique).

Ainsi, les développeurs sont libres de choisir le processus qui leur semble le plus adapté en fonction du type d'applications développées, de leurs habitudes ou encore de leur niveau de maturité objet.

Toutes les phases d'un projet peuvent être décrites suivant la notation unifiée. Dans le cadre de ce travail, UML est utilisé pour décrire les fonctionnalités du système. Le diagramme des composants, qui contient les différents *packages* de l'application est également utilisé.
Il faut surtout retenir que UML est un langage universel qui permet à chaque acteur de réagir et de s'exprimer suivant son propre point de vue et être compris de tous les autres acteurs.

5.2 Le serveur Tomcat

Tomcat est un containeur de servlet avec un environnement JSP. Un containeur de servlet est un *runtime shell* qui gère et invoque les servlets à la demande des utilisateurs.
Il est possible d'utiliser Tomcat *stand-alone* ou avec un autre serveur Web comme Apache.
Il supporte les spécifications des Java Servlets 2.3 et JSP 1.2. C'est un des serveurs *open source* le plus répandu.

5.2.1 Installation de Tomcat

Pour installer Tomcat, il faut décompresser l'archive téléchargée de jakarta-tomcat [Apache 2002] à un emplacement approprié, comme `C:\webshop\.`
Pendant la configuration du système, il faut considérer les points suivants:

- Installer un kit de développement Java (JDK, pour Java Development Kit).
- Définir la variable d'environnement `JAVA_HOME` vers le chemin de l'installation JDK. Tomcat et `ant` (cf. 5.2.3) utilisent `JAVA_HOME` dans leurs scripts de démarrage afin de localiser les fichiers nécessaires comme `java.exe`, `javac.exe` et `tools.jar` (librairie qui contient les classes java).
- Définir la variable d'environnement `CATALINA_HOME` vers le chemin de l'installation de jakarta-tomcat.

Ces lignes ont été ajoutées au fichier `%CATALINA-HOME%\bin\catalina.bat`:

- `Set JAVA_HOME=C:\jdk1.3`
- `Set CATALINA_HOME=C:\webshop\jakarta-tomcat-4.0.1`

Il faut enfin exécuter le fichier *batch* `startup.bat` du répertoire `%CATALINA-HOME%\bin`. Le résultat peut être contrôlé en se rendant sur `http://localhost:8080` avec le navigateur. Le script `shutdown.bat` est exécuté pour éteindre le serveur.

Les applications Web sont enregistrées dans le répertoire `%CATALINA-HOME%\webapps`. Chaque application possède un fichier `web.xml` qui contient sa configuration.

5.2.2 Les fichiers de configuration

Il convient d'analyser le fichier de configuration du serveur, `serveur.xml` et le fichier de configuration de l'application, `web.xml`.

Server.xml

Le serveur Tomcat est configuré à l'aide du fichier `server.xml` situé dans le répertoire `%CATALINA-HOME%\conf`.
La balise `<Context>` permet de modifier quelques attributs par rapport aux valeurs par défaut. Par exemple, `docBase` positionne la racine des documents de l'application Web à un autre point que `%CATALINA_HOME%/webapps/` répertoire.

Web.xml

Le fichier `WEB_INF\web.xml` procure la configuration de l'application Web.
Il existe également un de ces fichiers pour la configuration de niveau serveur dans `jakarta-tomcat/conf`.
Ce fichier définit les éléments suivants :

- Nom affiché pour l'introduction générale à l'application Web;
- paramètres de contexte des informations sur l'application déployée, par exemple l'adresse électronique de l'administrateur système;
- noms de servlets et correspondances;
- configuration de session;
- correspondances des bibliothèques de balises (établit le lien entre URL de la page et le fichier `*.tld` réel);
- types MIME pris en charge;
- liste de fichiers de bienvenue (les noms des pages par défaut qui, si elles sont présentes, sont chargées lorsque l'URL ne correspond qu'à un répertoire).

Chaque bibliothèque de balises possède un fichier `*.tld` décrivant les balises, leurs classes, le type de contenu du corps et leurs attributs. Ce fichier doit être accessible au conteneur JSP lorsque celui-ci rencontre une directive JSP `taglib`. La correspondance de la directive `taglib` au fichier actuel est représentée dans la figure 5.1.

```
<taglib>
    <taglib-uri>/tags/struts-html</taglib-uri>
    <taglib-location>/WEB-INF/taglibs/struts-html.tld</taglib-
          location>
</taglib>
```

Figure 5.1: correspondance de la directive `taglib` au fichier actuel.

5.2.3 Utilisation de ant et build.xml

Le serveur Tomcat est accompagné d'un outil, `ant`, qui aide à construire le projet et à placer tous les fichiers aux bons endroits. Le fichier `build.xml` décrit les pas nécessaires.
Ce fichier `build.xml` est placé à la racine des dossiers sur lesquels il va travailler, si bien que `basedir` est fixé au répertoire courant.

5.3 Java

5.3.1 Le langage Java

Depuis l'introduction de Java à la fin 1995, Java a conquis l'Internet rapidement. Il possède quelques propriétés qui le laissent apparaître comme langage de programmation idéal pour le monde bien câblé.

Java est un langage simple, orienté objet, distribué, interprété, robuste, sûr, neutre en architecture et portable, de grande puissance, *multithread*, dynamique.

- Java est un langage de programmation **orienté objet**. Les objets sont des instances des classes. Une classe est composée de champs (attributs) et de méthodes. Les méthodes interagissent avec les attributs. Les champs et les méthodes décrivent l'état et le comportement d'un objet. Les classes sont disposées dans une hiérarchie telle, que les sous-classes héritent du comportement des classes supérieures.
Java met un grand nombre de classes à disposition. Elles sont arrangées dans des paquetages qu'on utilise dans des programmes propres. Le paquetage `java.io` par exemple, s’occupe des entrées et des sorties.
Presque tout en Java est un objet, seules les primitives numériques et booléennes ainsi que les caractères sont des exceptions.
Une classe est l'unité de base de la compilation et de l'exécution. Tous les programmes Java sont des classes.
- Java est un langage **interprété** : Le compilateur ne produit pas directement du code machine, mais du "byte-code" pour la "Java Virtual Machine" (JVM). Lors de l'exécution d'un programme la JVM interprète le "byte-code". Comme le "byte-code" de Java est indépendant de la plate-forme, il est possible d'exécuter un programme Java sur un n’importe quel système où une JVM est implémenté.
Cette indépendance d'architecture est importante pour des applications qui sont distribuées par Internet ou d'autres réseaux.
- Java est un langage **simple**. Les développeurs ont essayé de créer un langage qu'un programmeur apprend rapidement, raison pour laquelle le nombre de termes du langage est petit. En outre les termes sont souvent les mêmes qu’en C. Une grande simplification est la suppression d’une grande source d'erreur - les pointeurs. Il y a aussi un *garbage collector*, qui enlève de la mémoire les variables qui ne sont plus utilisées.
- Java est **robuste** : avec `try`, `catch` et `finally` on peut regrouper le traitement d'exception (une exception est une erreur) sur une petite partie du code.
Java facilite l'utilisation des *threads* en les supportant directement dans le langage. Le paquetage `java.lang` met à disposition la classe `Thread` qui contient des méthodes pour la manipulation des processus.
Java supporte aussi l'utilisation de plusieurs *threads* qui exécutent différents travaux en même temps (*multithread*).

La figure 5.2 représente le programme Java le plus simple "HelloWorld" qui affiche le message "Hello World!" à l’écran.

```
public class HelloWorld {
      public static void main (String[] args) {
            System.out.println("Hello World!");
      }
}
```

Figure 5.2: l'application `HelloWorld`.

5.3.2 JavaBeans

Un "JavaBean" est un composant réutilisable et affichable à l'aide d'un outil générique. Tout objet qui respecte quelques règles peut être un "bean". Mais il n'y a pas de classe "bean" de laquelle tout "bean" devrait être une sous-classe.
Un composant doit tout simplement implémenter l'interface `java.io.Serializable`.
La figure 5.3 montre l'exemple d'un bean très simple. "SimpleBean" possède le champ `theName` et les méthodes qui permettent d'interagir avec celui-ci.

```
public class SimpleBean implements java.io.Serializable  {
      protected string theName;

      public SimpleBean() {
      }
      public void setTheName (string newName) {
            theName = newName;
      }
      public String getTheName {
            return theName;
      }
}
```

Figure 5.3 : code source de `SimpleBean.java`.

5.4 JSP

Une page JSP (*JavaServer Pages*) est un fichier texte combinant du HTML standard et de nouvelles balises de script. Les pages JSP ressemblent à du HTML, mais elles sont compilées en servlets Java lors de leur première invocation. La servlet qui en résulte est une combinaison du HTML provenant du fichier JSP et du contenu dynamique incorporé spécifié par les balises.

Les éléments d'une page JSP se répartissent en deux catégories:

- Les éléments traités sur le serveur.
- Les modèles de données ou toute autre chose ignorée par le moteur traitant la page JSP.

Une page JSP est exécutée par un moteur JSP ou conteneur, installé sur un serveur Web ou sur un serveur d'application. Lorsqu'un client demande une ressource JSP, le moteur enveloppe cette requête et la transmet à JSP en compagnie d'un objet réponse. JSP traite la requête et modifie l'objet réponse pour y incorporer la communication avec le client. Le conteneur enveloppe alors les réponses de la page JSP et les envoie au client. La couche sous-jacente à JSP est celle l'implémentation d'une servlet.

La requête et la réponse ont les mêmes abstractions que `javax.servlet.http.HttpServletRequest` ou `javax.servlet.http.ServletResponse`.

La première fois que le moteur intercepte une requête pour une page JSP, il compile cet élément de traduction (la page JSP et les autres fichiers dépendants) dans un fichier de classe implémentant le protocole de servlet. Si les fichiers dépendants sont eux-mêmes des JSP, ils sont compilés dans leurs propres classes.

La classe de servlet générée à la fin du processus de traduction doit s'étendre en une superclasse qui est:

- Spécifiée par l'auteur du JSP grâce à l'attribut `extends` de la directive `page`.
- Une classe d'implémentation spécifique au conteneur JSP implémentant l'interface `javax.servlet.jsp.JspPage` et procurant des comportements propres à la page. Comme la plupart des pages JSP utilisent `http`, les classes d'implémentation doivent implémenter l'interface `javax.servlet.jsp.HttpJspPage`, une sous-interface de `javax.servlet.jsp.JspPage`.

L'interface `javax.servlet.jsp.JspPage` contient deux méthodes:

- `Public void jspInit().`
- `Public void jspDestroy().`

L'interface `javax.servlet.jsp.HttpJspPage` contient la méthode suivante:

```
public void_jspService(HttpServletRequest request,
      HttpServletResponse response)throws ServletException,
      IOException
```

Cette méthode correspond au corps de la page JSP et est utilisée pour traiter les requêtes en *threads*. L'implémentation de cette méthode est générée par le conteneur JSP et ne devrait jamais être le fait de l'auteur de la page.

Chaque fois qu'une requête parvient à la page JSP, la méthode `_jspService` est invoquée, la requête est traitée, puis la page JSP génère la réponse appropriée. La réponse est récupérée par le conteneur et renvoyée au client.

5.4.1 La syntaxe JSP

Les URL utilisées par les pages JSP respectent les conventions des servlets: une URL débutant par "/", ce que l'on nomme chemin relatif au contexte, est interprété par référence à l'application ou au `ServletContext` auquel appartient la page JSP. Si l'URL ne débute pas par "/", elle est interprétée en relation avec la JSP active.

5.4.2 Les éléments d'une page JSP

Les différents éléments d'une page JSP peuvent être répartis dans les catégories suivantes:

- Les directives,
- les éléments de script (déclarations, scriptlets et expressions),
- les actions standards.

Les directives

Les directives JSP servent de messages envoyés par la page JSP au conteneur JSP. Elles sont utilisées pour définir des valeurs globales comme des déclarations de classe, des méthodes à

implémenter, un type de contenu de sortie, etc. Elles ne produisent aucune sortie vers le client. Leur syntaxe générale est:
`<% nomdirective attribut="valeur" attribut="valeur"%>`
Les trois directives sont:

- La directive `page`,
- la directive `include`,
- la directive `taglib`.

La directive `page` – Elle définit plusieurs attributs importants affectant la totalité de la page.

La directive `include` – Elle indique au conteneur d'inclure le contenu de la ressource dans la page JSP active, en ligne, à l'endroit indiqué. Le fichier spécifié doit être accessible et disponible pour le conteneur JSP.

La directive `taglib` – Elle permet à la page d'utiliser des balises personnalisées définies par l'utilisateur. Elle nomme aussi la bibliothèque de balises (un fichier compressé) renfermant les définitions de celles-ci.

Les éléments de script

Les éléments de script servent à placer du code de script (le plus souvent du code Java) dans la page JSP. Ils permettent de déclarer des variables et des méthodes, d'incorporer du code de script arbitraire et d'évaluer une expression.
Il existe trois types d'éléments de script:

- Les déclarations,
- les scriptlets,
- les expressions.

Les déclarations – Une déclaration est un bloc de code Java placé dans une page JSP et utilisé pour définir des variables et des méthodes avec leur portée de classe dans le fichier de classe généré. Tout ce qui est défini dans une déclaration est disponible pour toute la page JSP. Un élément de déclaration est encadré par les balises `<%! et %>`.

Les scriptlets – Une scriptlet est un bloc de code Java exécuté au moment traitement de la requête. Elle est encadrée par des balises `<% et %>`.

Les expressions – Une expression est une abréviation correspondant à une scriptlet envoyant une valeur dans le flux de réponse vers le client. Lors de l'évaluation de l'expression, le résultat est converti en une chaîne puis affiché.
Une expression est encadrée par des balises `<%= et %>`. Si une partie quelconque de l'expression est un objet, la conversion s'effectue à l'aide de la méthode `toString()` de l'objet.

Les actions standards

Les actions sont des balises spécifiques qui affectent le comportement du script JSP lors de son exécution ainsi que les réponses envoyées au client. La spécification JSP dresse la liste de quelques types d'actions standard, devant être fournies par tous les conteneurs quelle que soit l'implémentation. Les actions standards procurent aux auteurs des pages des fonctionnalités de base. Voici la liste des actions standards:

- `<jsp:useBean>;`
- `<jsp:setProperty>;`

- <jsp:getProperty>;
- <jsp:param>;
- <jsp:include>;
- <jsp:forward>.

Les balises <jsp:useBean>, <jsp:setProperty> et <jsp:getProperty> sont utilisées conjointement avec les JavaBeans. Les JavaBeans sont des composants logiciels (des classes Java) pouvant servir à encapsuler du code Java pour séparer la présentation du contenu à l'intérieur des pages JSP.

Jsp:useBean – Une action jsp:useBean permet d'associer un JavaBean au script JSP.
En interne, le conteneur cherche d'abord à localiser l'objet à l'aide de l'identifiant et du scope et, s'il ne le trouve pas, utilise le nom du bean comme argument de la méthode instantiate() de java.beans.Beans en se servant du gestionnaire de classe actif. Si l'instanciation échoue, une exception est générée par instantiate() au moment de la requête. La figure 5.4 représente la syntaxe d'une action jsp:useBean.

```
<jsp:useBean id="nom" scope="page|request|session|application"
beandetails />
```

Figure 5.4: syntaxe d'une action jsp:useBean.

JSP:setProperty – Cette action standard est utilisée en conjonction avec l'action useBean. Elle fixe la valeur des propriétés d'un bean. Sa syntaxe est représentée dans la figure 5.5.

```
<jsp:setProperty name="nomBean" propriétédétails/>
```

Figure 5.5: syntaxe d'une action jsp:setProperty

Propriétédétails est l'une des valeurs suivantes:

- property= "*";
- property= "nomPropriété";
- property= "nomPropriété" param= "nomParamètre";
- property= "nomPropriété" value= "valeurPropriété";

ValeurPropriété est une chaîne ou une scriptlet.

jsp:getProperty – L'action jsp:getProperty est complémentaire de l'action jsp:setProperty et permet d'accéder aux propriétés d'un bean. Elle accède à la valeur d'une propriété, la convertit en chaîne, qu'elle envoie dans le flux de sortie.

Pour convertir la propriété en chaîne (String), la méthode toString() est invoquée sur la valeur de la propriété s'il s'agit d'un objet ou convertie directement s'il s'agit d'une valeur primitive.
La syntaxe générale de l'action jsp:getProperty est représentée dans la figure 5.6.

```
<jsp:getProperty name="nom" property="nompropriété"/>
```

Figure 5.6: syntaxe d'une action jsp:getProperty

`jsp:param` – L'action `jsp:param` est utilisée afin de procurer aux autres balises des informations complémentaires sous la forme de couples nom-valeur. Cette action est utilisée en combinaison avec les actions `jsp:include`, `jsp:forward`.

`jsp:include` – Cette action permet d'inclure une ressource statique ou dynamique dans le script JSP actif au moment de la requête. Sa syntaxe est représentée dans la figure 5.7.

```
<jsp:include page="nomFichier" flush="true"/>
ou
<jsp:include page="urlSpec" flush="true">
      <jsp:param name="nomParam" value="valeurParam" />
      …
</jsp:include>
```

Figure 5.7: syntaxe d'une action `jsp:include`

`jsp:forward` – L'action `jsp:forward` permet de transmettre la requête à un autre script JSP, à une servlet ou à une ressource statique. Sa syntaxe est représentée dans la figure 5.8.

```
<jsp:forward page="url" />
ou
<jsp:forward page="urlSpec">
      <jsp:param name="nomParam" value="valeurParam" />
</jsp:forward>
```

Figure 5.8 : syntaxe d'une action `jsp:forward`

5.5 Définition de Struts

Struts [Struts 2002] est un *framework* sous licence *open source* servant à développer les applications Web dynamiques.

Dans une application Web dynamique, une page est générée au moment de la requête de l'utilisateur. Souvent le contenu de la page provient d'une base de données.

Struts utilise deux technologies opérant sur les serveurs pour générer les pages Web dynamiques: Java Servlets et JavaServer Pages

5.5.1 Les Servlets

Les servlets sont la réponse technologique de Java à la programmation CGI (*Common Gateway Interface*). Ils sont plus efficients, plus puissants et plus portables que les scripts CGI traditionnels.

Au niveau de la programmation, une servlet ressemble à une applet côté serveur. Il s'agit effectivement d'un exécutable écrit en Java et qui est habituellement exécuté en réponse à une invocation depuis une page HTML.

5.5.2 JSP

La technologie JavaServer Pages (cf. section 5.4) est une extension de la technologie Java Servlets.
Quand un utilisateur invoque une page JSP, le navigateur envoie une requête au serveur Web qui passe le contrôle au moteur JSP. Le moteur JSP contrôle si cette page a été compilée en

une servlet. Si tel n'est pas le cas ou si la page JSP est plus récente que la servlet (spécifiquement le fichier de la classe qui représente la servlet), le moteur traduit et compile la page JSP pour créer une servlet et lui passer la requête. La servlet génère la réponse qui est retournée au navigateur par le serveur Web.

5.5.3 Architecture d'application

L'architecture d'une application Struts est basée sur le *pattern* Modèle-Vue-Contrôleur.
Modèle-Vue-Contrôleur (MVC) est le concept introduit par les inventeurs de Smalltalk pour encapsuler certaines données ensembles avec leur traitement (le modèle: les données se trouvent dans `ActionForm` (`CVForm`, `InscriptionForm`) et sont traiter avec la classe `Action`). La classe `Action` est étendue pour faire une action spécifique pour une requête URI. Elle exécute la logique métier (sauvegarder ou chercher les données) et l'isole de l'interaction avec l'utilisateur (contrôleur) et la présentation de ces données (la vue). MVC fournit un mécanisme pour lier un modèle, les vues d'une façon standardisée de telle sorte que le modèle communique les changements dans son état à toutes les vues qui y sont rattachées. Un changement dans l'état d'un modèle survient soit parce que le contrôleur a donné une quelconque instruction, soit pour une raison interne.

Plus simplement, les composants de l'architecture MVC interagissent comme montré sur la figure 5.9.

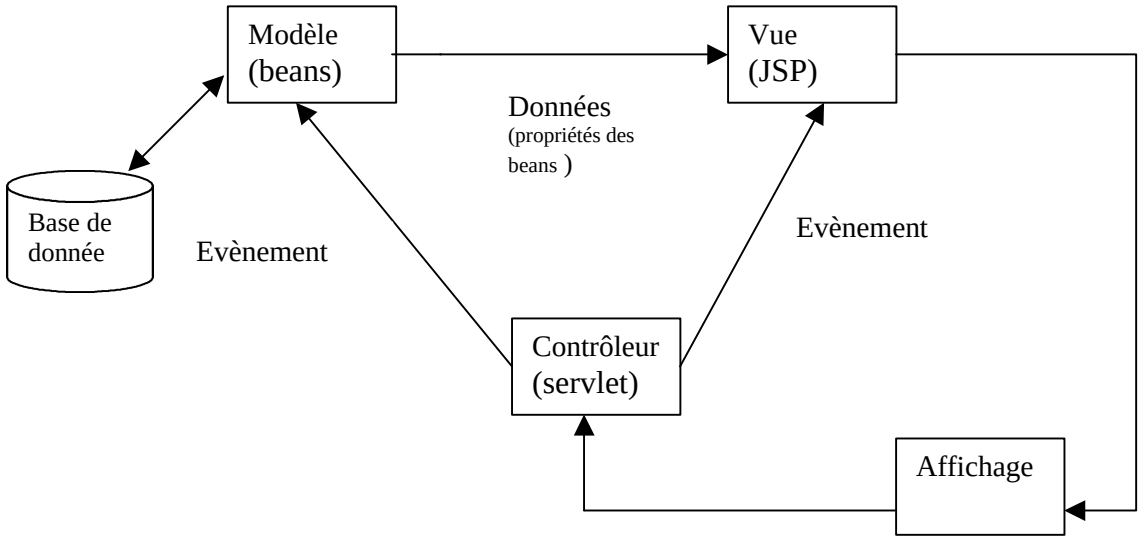


Figure 5.9: architecture MVC

Le modèle MVC utilise une servlet pour recevoir la requête. La servlet délègue la collecte de données pour la requête à un JavaBean. Le JavaBean collecte les données nécessaires pour satisfaire la requête en faisant appel aux bases de données et retourne ensuite le contrôle au contrôleur. Le contrôleur dirige alors la requête à la JSP qui construit la réponse HTML utilisant les données du JavaBean et son propre code HTML. Après la construction, la réponse est envoyée au navigateur pour l'affichage.

En général, les composants d'une application basée sur Struts sont les suivants:

- Les formulaires HTML générés par les JSP utilisant la librairie de *tag* html Struts.
- L'`ActionServlet` contrôleur auquel le formulaire de données est posté. L'`ActionServlet` est configuré par un fichier XML, `struts-config.xml`.

- Les sous-classes spécifiques de l'application, de la classe `ActionForm` qui sont des JavaBeans avec les propriétés correspondant aux champs du formulaire. `ActionServlet` instancie automatiquement le bean de formulaire, remplit ses propriétés avec les données du formulaire et le stocke dans une portée spécifiée (généralement la requête ou la session).
  L'`ActionServlet` peut alors optionnellement demander à la sous-classe de `ActionForm` de valider son contenu pour contrôler que les données reçues de l'utilisateur sont valides. Dans le cas contraire, l'`ActionServlet` redirige le contrôle au formulaire JSP original. La librairie de *tags* html inclut l'habileté pour le JSP d'afficher les messages d'erreur appropriés et de remplir le formulaire avec les données précédentes.
- Les sous-classes spécifiques de l'application, de la classe `Action`.Les sous-classes de la classe `Action` contiennent la logique métier de l'application dans la méthode `perform()`. Par cette méthode, il est possible d'examiner le formulaire de données, d'exécuter le traitement approprié et de stocker les données nécessaires dans la requête ou la session qui seront utilisées par le JSP pour produire la réponse.
  La méthode `perform()` retourne un `ActionForward`, une classe Struts qui représente le JSP qui sera utilisé pour produire la réponse. Le *mapping* entre les noms logiques et les actuels JSP sont établis dans le fichier `struts-config.xml`.
- L'`ActionMapping` est une classe Struts définie représentant le *mapping* entre un *pattern* URL utilisé dans une requête client et les sous-classes d'`ActionForm` et d'`Action` qui seront utilisées.
- La base de données de l'application Web.

Les composants de Struts sont expliqués en détail dans les lignes qui suivent.

Les formulaires HTML et les JSP

Une page JSP Struts utilise des tags Struts, et, moins fréquemment, des tags JSP pour extraire des données au moyen de déclarations, à partir d'attributs de portée (objets ou JavaBeans). Ces objets ont été placés dans une portée par les objets d 'action, habituellement comme résultats d'un traitement de logique métier.
Les JavaBeans sont utilisés pour gérer les formulaires de données. Un problème-clé dans la conception d'applications Web est la rétention et la validation de ce que l'utilisateur a entré. Avec Struts on peut facilement entreposer les données d'un formulaire JSP dans un bean de formulaire. Le bean est sauvegardé en tant qu'attribut dans une des collections standards partageant un contexte, de telle sorte qu'il puisse être utilisé par d'autres objets, spécialement les objets d'action validant (par exemple la méthode `perform` de la classe `Action` ou `validate` de la classe `ActionForm`) les données d'utilisateur. Dans le cas où il y aurait des erreurs de validation, le bean de formulaire est utilisé par la JSP de nouveau pour repeupler les champs du formulaire et afficher les messages d'erreur.

ActionServlet

C'est le contrôleur MVC qui joue un rôle de distributeur d'actions.
Sa fonction est de recevoir une requête de client (typiquement, un utilisateur se servant d'un navigateur Web). L'`ActionServlet` délégue la requête à un gestionnaire d'action approprié qui exécute une fonction de logique métier encapsulée dans le modèle, et transmet ensuite le contrôle à la JSP ou à l'action suivante.
Une application Web Struts possède un seul contrôleur, mais plusieurs actions, JSP et JavaBeans. Le contrôleur est une servlet compilée et chargée au démarrage du serveur d'application, à partir de sa définition dans le fichier de déploiement d'application XML.

Le même fichier contient les déclarations des librairies de *tags* Struts et le *mapping* du servlet contrôleur, indiquant quels types de requête d'utilisateur vont invoquer la servlet.

Bien qu'il n'existe qu'une seule servlet contrôleur d'application, pour chaque nouvelle requête une nouvelle tâche de servlet (*threads*) est créée. Comme les tâches nécessitent moins de ressources que les processus de système d'exploitation créés en programme CGI pour chaque requête d'utilisateur, les servlets sont beaucoup plus performants que les processus CGI. Les différentes tâches d'une servlet partagent les mêmes propriétés de servlets, mais ont des variables de méthode spécifiques. Si une propriété d'une servlet est utilisée dans une application Web, la synchronisation exigeante doit être faite pour prévenir les conflits.

Au moment de son initialisation, le contrôleur lit le fichier de configuration XML de l'application (`WEB_INF/web.xml`) contenant les déclarations de beans de formulaire et les *mapping* d'actions. Quand un utilisateur effectue une requête avec un certain *pattern*, habituellement depuis une JSP, la requête est envoyée au contrôleur, qui est prêt pour l'action. Le contrôleur trouve la définition d'action et vérifie si l'objet d'action a déjà été créé. Si ce n'est pas le cas, il le crée. Si oui il utilise l'objet d'action déjà créé.

Avant de déléguer la requête d'utilisateur à l'objet d'action, la servlet contrôleur peut effectuer un certain travail. Si une action possède un bean de formulaire, le contrôleur va copier tous les paramètres de la requête dans le bean de formulaire en se basant sur les noms identiques des paramètres-propriétés.
Si la validation des donnés du formulaire est requise par le paramètre de servlet dans le fichier de déploiement de l'application, le contrôleur va appeler la méthode `validate` du bean de formulaire.
S'il y a au moins une erreur et que le *tag* d'erreur est utilisé dans la page JSP contenant le formulaire d'insertion des données, le contrôleur va regrouper les messages d'erreur et les afficher dans la même page JSP avec les valeurs des propriétés du bean de formulaire en tant que valeur de champ, de telle sorte qu'un utilisateur puisse changer les valeurs invalides.

Ensuite, le contrôleur va déléguer la requête d'utilisateur à l'objet d'action plus précisément à la méthode `perform` de l'objet considéré avec quatre paramètres : *mapping* d'action, bean de formulaire, requête d'utilisateur et réponse à la requête.
A partir du paramètre d'action de la requête, l'action spécifique est déterminée, et, par conséquent, le traitement d'action approprié est effectué à l'aide des JavaBeans du modèle. A la fin du processus, la requête est transmise à l'action ou la JSP suivante.

Les objets d'action sont liés au contrôleur de l'application et ont donc accès aux méthodes de cette servlet. En transmettant un contrôle, un objet d'action peut indirectement transmettre un ou plusieurs objets partagés, incluant des JavaBeans en plaçant ceux-ci en tant qu'attributs dans une des collections standards partagées par les servlets et les JSP. Chaque collection possède différentes règles pour la durée de vie de la collection en question et la visibilité des objets qui y sont entreposés.

Ensemble, les règles définissant la durée de vie et la visibilité sont appelées la portée d'attribut (*attribute scope*) :

- Page: attributs qui sont visibles à l'intérieur d'une seule page JSP, pour la durée de vie de la requête courante.
- Requête: attributs qui sont visibles à l'intérieur d'une seule page JSP, aussi à l'intérieur de n'importe quelle servlet ou page étant incluse dans cette page; il peut

aussi s'agir d'une servlet ou d'une page vers laquelle la page courante aurait effectué un transfert (*forward*).

- Sessions: attributs qui sont visibles pour tous les servlets et pages JSP qui participent à une session d'utilisateur particulière, à travers une ou plusieurs requêtes.
- Application: attributs qui sont visibles pour tous les servlets et pages JSP qui font partie d'une application Web.

Les sessions sont créées et gérées par le contrôleur pour chaque nouvelle requête d'utilisateur. Par défaut Struts utilise des *cookies* pour reconnaître une session d'utilisateur. Si un utilisateur interdit les *cookies*, la réécriture d'URL avec un identifiant de session est faite.

ActionMapping

Cette classe représente un *mapping* entre un pattern URL et un composant de logique métier (`Action`). A l'initialisation le contrôleur analyse un fichier de configuration. La configuration définit (entre autres choses) les *mapping* d'action pour l'application. Au minimum un *mapping* doit préciser un chemin de requête et le type de l'objet d'action qui doit agir lors de la requête. Le contrôleur utilise les *mapping* d'action pour transformer les requêtes http en actions d'application et en réponse JSP.

ActionForm

C'est un composant du modèle MVC. Dans Struts, ce composant opère comme un JavaBean qui représente les données du formulaire de la page JSP. Les JavaBeans sont des composants réutilisables qui sont des classes Java avec quelques restrictions additionnelles. Au minimum, un JavaBean doit avoir un constructeur, par défaut sans argument, et, pour chaque propriété, des méthodes `get` et `set`. L'`ActionServlet` remplit automatiquement les propriétés du bean (champs du formulaire) après l'instanciation. Si le bean a une méthode `validate()`, cette méthode est appelée avant l'appel de la classe `Action`. L'`ActionForm` est étendu pour représenter une vue JSP.

Action

Cette classe est un composant du modèle MVC et représente la logique métier. Cette classe est étendue pour implémenter la logique métier d'une requête particulière. Son interaction avec le autres composants est décrit dans le paragraphe ActionServlet.

Taglib

Ce composant représente toutes les librairies de *tags* utilisées dans Struts: bean, html, logic, et template.

Les librairies de *tags* Struts sont utilisées par les vues JSP. Il y a quatre librairies de tags dans Struts 1.0:

- La librairie html contient des *tags* permettant de générer le langage HTML. Elle permet d'afficher des valeurs de l'`ActionForm` dans les champs d'entrée HTML. Elle encode les liens de l'URL avec la session de sorte que des *cookies* ne soient pas exigés par le *browser* du client.
- La librairie bean est concernée par la manipulation des JavaBeans dans les JSP. Cette librairie contient des *tags* qui définissent de nouvelles variables de script basées sur les propriétés des beans existants.
- La librairie logic est utilisée pour contrôler le flux dans une vue JSP.
- La librairie template permet de construire des pages JSP utilisant un format commun de templates dynamiques.

Les *tags* Struts habituels sont conçus pour utiliser les caractéristiques d'internationalisation de Java. Les libellés de champs et les messages peuvent être puisés dans un fichier-ressource de messages.
Pour le pays et la langue d'un client, Java peut automatiquement fournir les ressources appropriées à l'aide d'un fichier de message approprié (`ApplicationResources_fr.properties`).
La balise `<bean:message key="cv.create"/>` affiche le texte de la clé `cv.create`.

Base de données du modèle

Dans une application utilisant une base de données, les beans de logique métier se connectent et interrogent une base de données. Ils retournent les résultats à l'action afin d'être entreposés dans un bean de formulaire et ensuite affichés dans une JSP.
Une source de données à une base de données relationnelle est définie dans le fichier de configuration XML de l'application. Struts s'occupe de la gestion de la connexion pool de la source de données (le fichier `poolman.xml` qui se trouve dans le répertoire `WEB-INF\classes`).

Connexion Pool

Dans des applications Web dynamiques puisant le contenu des pages dans une base de données, l'établissement d'une connexion à une base de données est un processus très exigeant en ressources et en temps. Dans une application client-serveur, un client a habituellement une connexion qui lui est dédiée pour la durée de la session de client. Cela restreint le nombre de clients, puisque le système de gestion de base de données serveur (SGBD) ne peut prendre en charge de façon efficiente qu'un certain nombre de connexions ouvertes actives au même moment. Par conséquent, ce type d'application est plus approprié pour un Intranet. Une application Web peut avoir plusieurs clients.

Dédier une connexion à une base de données à une seule session client n'est pas une option viable puisqu'une session peut durer assez longtemps. Produire une nouvelle connexion pour chaque requête de client et fermer la connexion à la fin du traitement de la requête réduit le nombre de connexions parallèles. Cependant la production d'une nouvelle connexion nécessite du temps, entraînant une baisse de performance de l'application. C'est la raison pour laquelle une connexion pool est utilisée. Un nombre prédéfini de connexions sont créées au moment de l'initialisation de l'application et une nouvelle requête obtient de façon efficiente une connexion disponible du pool, qui est retournée au pool aussitôt qu'elle n'est plus nécessaire dans le traitement de la requête.

5.5.4 Installer Struts

Il faut télécharger Struts dans un répertoire quelconque nommé `%STRUTSHOME%`. Ensuite il faut copier les fichiers `*.jar` qui se trouvent dans `%STRUTSHOME%/lib` dans le répertoire `WEB-INF/lib` de l'application Web. Les bibliothèques de balises qui se trouvent dans le même répertoire peuvent être placées dans un dossier quelconque. L'emplacement doit être défini dans le fichier de configuration `web.xml`. La figure 5.10 contient un exemple de description d'une librairie de balises.

```
<taglib>
    <taglib-uri>/tags/struts-bean</taglib-uri>
         <taglib-location>/WEB-INF/taglibs/struts-bean.tld
     </taglib-location>
</taglib>
```

Figure 5.10: description d'une librairie de balises.

Struts contient un certain nombre d'applications Web et leur documentation (`struts-documentation.war`), un exemple d'application (`struts-example.war`), des démonstrations des librairies de *tags* et des dispositifs de téléchargement de fichier (`struts-template.war et struts-upload.war`) ainsi qu'une application Web vide (`struts-blank.war`).

6 Implémentation

6.1 Concept

L'idée est de mettre en place une implémentation simple permettant des développements ultérieurs. Dans le but de séparer la logique, la présentation et les données, le *framework* Struts a été choisi. Ce *framework* implémente le modèle MVC.
Grâce à cette technique de séparation de la logique et du contenu, un utilisateur avec peu de connaissances en informatique peut changer facilement l'interface graphique de l'application Web.
Les caractéristiques d'internationalisation Java ont été utilisées. En effet, Java offre un moyen simple de présenter un texte dans une autre langue, en se basant sur les préférences définies par l'utilisateur. Cependant, dans le cadre de ce travail, nous nous sommes limités au français.
Pour sauvegarder les données, une base de données relationnelle a été choisie.

L'application est destinée à être indépendante de la plate-forme. Par conséquent, la programmation est effectuée en Java et le serveur Tomcat 4.0 qui supporte les JSP 1.2 et les servlets 2.3 a été choisi.
En ce qui concerne le déploiement, `ant` et le fichier `build.xml` ont été choisis.
Pour l'authentification et les droits d'accès, le mécanisme de Tomcat est utilisé. Les fichiers à droits d'accès restreints ont été placés dans les répertoires protégés par Tomcat.

Une requête dans la base de donnée est lente. Dans le but de diminuer le temps de chargement d'une page `*.jsp`, les données souvent utilisées sont sauvegardées dans un bean de session. C'est pour cette raison que la première action invoquée après qu'un utilisateur soit logué est la création du bean utilisateur (`Candidat` ou `Recruteur`).

L'emplacement des fichiers de l'application jobplace est défini comme suit:

- Les JSP se trouvent dans le répertoire `/jobplace/pages` ou dans un de ses sous-répertoires.
- Le CLASSPATH des beans et autres classes de prise en charge Java débutent dans le répertoire `/jobplace/WEB-INF/classes`, c'est là que doivent démarrer les noms des packages.
- Les fichiers JAR des classes de prise en charge doivent se trouver dans le répertoire `/jobplace/WEB-INF/lib`, où elles seront ajoutées au CLASSPATH.
- Les fichiers descripteurs de bibliothèques de balises sont situés dans `/jobplace/WEB-INF/taglibs`.
- Les fichiers de configurations `*.xml` et `*.properties` se trouvent dans `/jobplace/WEB-INF` ou dans un de ses sous-répertoires.

Les parties de l'implémentation qui vont être examinées sont les suivantes:

- La base de données,
- le débogage et le *logging*,
- les fichiers de configuration,
- le contrôle d'accès,
- le design,
- les composants de l'application jobplace: les *packages*.

6.2 La base de données

Le système de recrutement en ligne implémenté s'appuie sur une base de données. L'objectif de ce sous-chapitre consiste donc à concevoir les tables à utiliser.
La conception de la base de données comporte trois phases : l'analyse des données, la construction du modèle entité-association et sa conversion en un schéma de base de données relationnelle.

Le logiciel MySQL [MySQL 2002] a été choisi pour la création de la base de données. Des commandes SQL ont été utilisées pour la création des tables et leur manipulation. Les transactions n'étant pas supportées par MySQL, il a fallu définir les tables de type BDB.
BerkeleyDB [Sleepycat 2002] propose un support pour des tables transactionnelles sous MySQL. En utilisant les tables BerkleyDB il est possible d'utiliser `COMMIT` et `ROLLBACK` sur les transactions.

Les étapes d'élaboration de la base de données vont être décrites dans les lignes qui suivent.

6.2.1 L'analyse des données

L'analyse des données vise à déterminer, en collaboration avec les utilisateurs, les données nécessaires au système d'information, leurs liaisons et leurs structures. Le modèle de cas d'utilisation (cf. chapitre 4) a servi de guide pour l'analyse de données en fournissant une meilleure compréhension des fonctionnalités du système et en définissant ses contours.

6.2.2 Le modèle entité-association

A partir du modèle de cas d'utilisation, il a été possible d'identifier les principaux ensembles d'entités et les ensembles de liens entre ces entités. La figure 6.1 représente le modèle entité-association du système de recrutement en ligne.
Une entité (*entity*) est un objet spécifique pouvant être identifié distinctement parmi d'autres objets, dans le monde réel ou dans notre pensée. Elle peut désigner une personne, un objet, un concept abstrait ou un événement. Les entités de même type forment un ensemble d'entités caractérisées par un certain nombre d'attributs.
Une association d'un ensemble d'entités EE_1 à un deuxième ensemble EE_2 définit un lien orienté dans cette direction.
On attribut un type à chaque association d'un ensemble d'entités EE_1 à un autre ensemble EE_2. Le type d'association de EE_1 à EE_2 indique le nombre d'entités provenant de l'ensemble associé EE_2 qui sont reliées à une entité dans EE_1. On distingue essentiellement quatre types d'associations : simple, conditionnel, multiple et multiple conditionnel.

Association simple (type 1)

Dans une association simple ou de type 1, à chaque entité de l'ensemble d'entités EE_1 correspond "une et une seule" entité dans l'ensemble EE_2. Par exemple dans notre analyse des données, chaque CV est possédé par un seul candidat. Par conséquent, dans la figure 6.1, cette association est de type 1.

Association conditionnelle (type c)

Dans une association conditionnelle ou de type c, à chaque entité de l'ensemble d'entités EE_1 correspond "aucune ou une", donc au plus une entité dans l'ensemble EE_2. Par exemple, dans la figure 6.1, un candidat possède ou non un seul CV.

Association multiple (type m)

Dans une association multiple ou de type m, à chaque entité de l'ensemble d'entités EE_1 correspondent "plusieurs" entités dans l'ensemble EE_2. Par exemple, dans la figure 6.1, un recruteur peut recevoir plusieurs CV correspondant à une offre d'emploi. Ce type d'association est dit complexe, car une entité dans EE_1 peut être reliée à un nombre quelconque d'entités dans EE_2.

Association multiple conditionnelle (type mc)

Dans une association multiple conditionnelle ou de type mc, à chaque entité de l'ensemble d'entités EE_1 correspondent "aucune, une ou plusieurs" entités dans l'ensemble EE_2. Par exemple, dans la figure 6.1, un candidat peut recevoir ou non des offres d'emploi correspondant à son profil. Ce type d'association est dit également dit complexe.

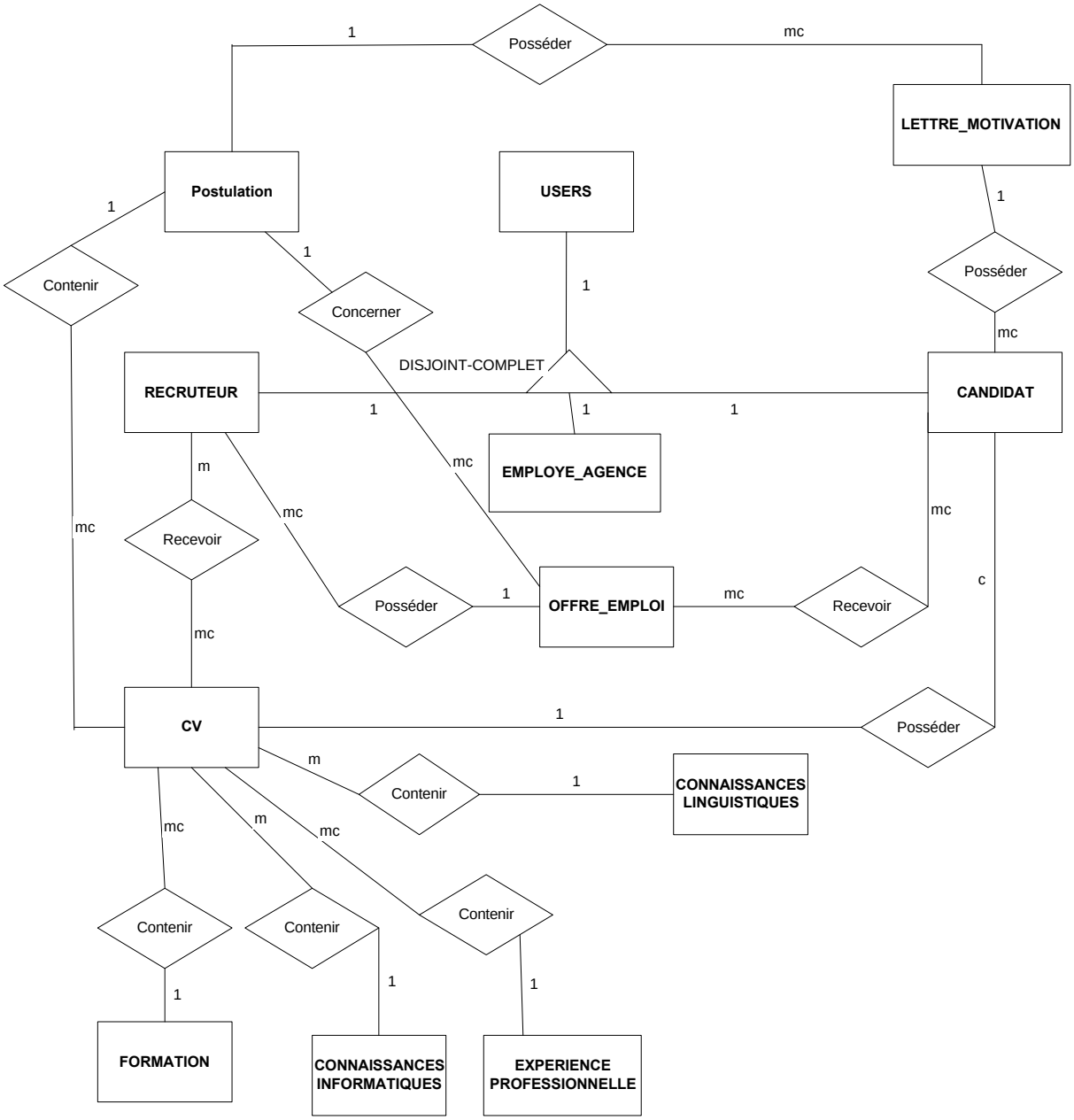


Figure 6.1: le modèle entité-association du système de recrutement en ligne.

Dans le modèle entité-association, les ensembles d'entités sont représentés graphiquement par des rectangles, et les ensembles de liens par des losanges.
L'ensemble d'entités Users généralise les ensembles d'entités Recruteur, Candidat et Employé d'agence.
Les sous-ensembles d'entités (Recruteur, Candidat, Employé d'agence) définis par spécialisation sont mutuellement disjoints, c'est-à-dire que leur intersection est vide. En effet, un utilisateur ou user ne peut appartenir qu'à une seule catégorie. Chaque utilisateur peut être soit recruteur, soit candidat, soit employé d'agence.
Lorsque les sous-ensembles d'entités définis par spécialisation sont mutuellement disjoints et contiennent tous les éléments de l'ensemble d'entités ascendant qui les généralise, on parle alors de "disjoint-complet".
Une hiérarchie de généralisation est représentée graphiquement par le symbole de la fourche accompagnée de la mention "disjoint-complet".

Il convient maintenant d'aborder la traduction du modèle entité-association en un schéma de base de données relationnelle. Il s'agit de définir une méthode permettant de représenter les ensembles d'entités et de liens par les tables.

6.2.3 Le schéma de base de données relationnelle

Un schéma de base de données (*database schema*) contient la description d'une base de données, plus précisément, la spécification de la structure des données et de leurs contraintes d'intégrité. Un schéma de base de données relationnelle contient la définition des tables, des attributs et des clés primaires (*primary-key*).

D'après [Meier 2001] la transformation du modèle entité-association en un schéma de base de données relationnelle est assurée par un certain nombre de règles:

- Chaque ensemble d'entités doit être traduit en une table distincte, avec une clé primaire unique.
- Chaque ensemble de liens peut être traduit en une table distincte qui doit contenir les clés d'identification des ensembles d'entités participantes comme clés étrangères. La clé étrangère (*foreign key*) dans une table est formée à partir d'un attribut ou d'une concaténation d'attributs qui sert de clé d'identification de la même table ou d'une table différente.
- Chaque ensemble de liens complexe-complexe doit être transformé en une table distincte. Les clés étrangères de la table sont formées au moins des clés d'identification des ensembles d'entités qui participent à la liaison. La clé primaire de la table est soit la clé d'identification formée par la concaténation des clés étrangères, soit une autre clé candidate. D'autres attributs de l'ensemble de liens deviennent les attributs de la tables.
- Un ensemble de liens simple-complexe peut s'exprimer dans les deux tables des ensembles d'entités participantes sans avoir besoin de créer une table distincte. Pour cela, dans la table qui participe à l'association simple (association de type 1 ou c) on définit une clé étrangère qui la relie à la table référencée en y ajoutant éventuellement d'autres attributs de l'ensemble de liens considéré.
- Un ensemble de liens simple-simple peut s'exprimer dans les deux tables des ensembles d'entités participantes sans avoir besoin de créer une table distincte. Une des clés d'identification de la table référencée est choisie comme clé étrangère dans la seconde table.
- Chaque ensemble d'entités dans une hiérarchie de généralisation donne lieu à une table distincte. La clé primaire de la table ascendante est aussi celle des tables au niveau inférieur.

A partir du modèle entité-association et des règles de passage au schéma de base de données relationnelle, les différentes tables peuvent être examinées. Dans la base de données, les ensembles d'entités sont représentés par les tables.
Pour chaque ensemble d'entités, une clé d'identification est définie. Une clé d'identification est un attribut ou une combinaison d'attributs qui permet de distinguer sans ambiguïté les entités dans l'ensemble considéré.

La base de donnée de l'application jobplace est créée dans MySQL. Le fichier `createtableJobplace.sql` contient les différentes tables de la base de données. Voici les requêtes SQL permettant de créer la base de données:

```
mysql>create database jobplace;
mysql>use jobplace;
mysql>source createtablesJobplace.sql;
```

La prochaine étape consiste à organiser le contenu de la base de données. Lorsqu'on crée une table, il faut lui attribuer un nom et spécifier le type de données à insérer dans ses colonnes. Les types utilisés dans le cas de jobplace sont: VARCHAR (pour les données du type chaîne de caractères), INTEGER (pour les données numériques) et DATE (pour les dates).
Dans les lignes qui suivent sont présentées les tables users et recruteur de la base de données jobplace. La description des autres tables se trouve en annexe A2.

La table users

La table users permet de sauvegarder le profil de l'utilisateur. Un utilisateur du système est caractérisé par quatre attributs : le nom (user_name), le mot de passe (user_pass), le rôle (user_role) et le numéro d'utilisateur (user_id).
Le numéro d'utilisateur permet de déterminer de manière univoque chacun des utilisateurs du système. Ce champ est défini comme identifiant principal de cette table (*primary key*).
Le rôle permet de faire une classification en indiquant le fait qu'un utilisateur est candidat, recruteur ou employé d'agence.

La requête SQL permettant de créer la table users est montrée dans la figure 6.2.

```
CREATE TABLE users (
     user_id integer not null auto_increment,
     user_name VARCHAR(25),
     user_pass VARCHAR(25),
     user_role VARCHAR(25),
     primary key(user_id))
     type=BDB;
```

Figure 6.2: requête SQL pour créer la table users.

La table recruteur

L'ensemble d'entités recruteur permet de sauvegarder le profil du recruteur. L'information demandée est le nom de la société, son secteur d'activité, son téléphone, son URL et sa description. Le titre, le nom, le prénom et l'email du contact sont aussi demandés. Le numéro d'utilisateur permet d'identifier de manière unique un recruteur.

La figure 6.3 présente la requête SQL pour créer la table recruteur.

```
CREATE TABLE recruteur (
      societe_nom VARCHAR(100),
      societe_secteur_activite VARCHAR(100),
      societe_phone VARCHAR(25),
      societe_url VARCHAR(25),
      societe_description VARCHAR(250),
      contact_titre VARCHAR(25),
      contact_nom VARCHAR(25),
      contact_prenom VARCHAR(25),
      contact_email VARCHAR(25),
      fk_user_id integer not null,
      foreign key(fk_user_id) references users)
      type=BDB;
```

Figure 6.3: requête SQL pour créer la table recruteur.

Le champ fk_user_id établit la relation entre la table recruteur et la table users. C'est une clé étrangère. Si un recruteur a un user_id de 40, alors le champ fk_user_id de ce recruteur sera également de 40. Ce champ permet également d'identifier de manière unique une occurrence de la table recruteur.

La table candidat

L'ensemble d'entités candidat permet de sauvegarder le profil du candidat. L'information demandée est le titre, le nom, le prénom, l'email, l'adresse (rue et numéro, ville, pays), la date de naissance, le téléphone fixe et le portable. Le numéro d'utilisateur permet d'identifier de manière unique un candidat.
Le champ fk_user_id établit la relation entre la table candidat et la table users. Si un candidat a un user_id de 30, alors le champ fk_user_id de ce candidat sera également de 30. Ce champ permet également d'identifier de manière unique une occurrence de la table candidat.

La table offre_emploi

En se basant sur les caractéristiques d'une offre d'emploi réel, les attributs suivants ont pu être identifiés : la date, le poste, la description du poste, les qualités requises, les connaissances techniques et le numéro d'identification.
Offre_emploi_id est un code d'identification. Ce champ doit contenir un nombre unique, afin de donner une désignation unique à la rangée de la table. Ce champ est défini comme identifiant principal de cette table.
Le champ fk_recruteur_id établit la relation entre le recruteur et l'offre d'emploi. Il s'agit d'une clé étrangère qui permet de savoir à quel recruteur appartient l'offre d'emploi.

La table cv

En se basant sur l'exemple d'un CV concret, l'ensemble d'entités CV a été subdivisé en plusieurs sous-ensembles : formation, expérience professionnelle, connaissances informatiques et connaissances linguistiques.
Ces tables ont toutes un identifiant secondaire (*foreign key*) faisant référence au CV auquel elles appartiennent.
La table CV proprement-dite est caractérisée par trois attributs : l'objectif, la date et la clé d'identification.
Le champ cv_id est un code d'identification. Ce champ doit contenir un nombre unique, afin de donner une désignation unique à la rangée de la table. Ce champ est défini comme identifiant principal de cette table.

Le champ fk_candidat_id établit la relation entre le candidat et le CV. Il s'agit d'une clé étrangère qui permet de savoir à quel candidat appartient le CV.

La table formation

L'ensemble d'entités formation est caractérisé par les attributs suivants : le nom de l'établissement, la durée de la formation, le diplôme obtenu, le lieu et le numéro de formation comme clé d'identification artificielle.

La table experience_professionnelle

L'ensemble d'entités expérience_professionnelle est caractérisé par les attributs suivants : le nom de l'entreprise, le secteur d'activité, le poste, la région, le début du travail, la fin du travail les et le numéro de l'expérience professionnelle comme clé d'identification artificielle.

La table connaissances_informatiques

L'ensemble d'entités connaissances_informatiques est caractérisé par les attributs suivants : les logiciels, les langages, les systèmes d'exploitation, la programmation et le numéro informatique comme clé d'identification artificielle.

La table connaissances_linguistiques

L'ensemble d'entités connaissances_linguistiques est caractérisé par les attributs suivants : le nom de la langue, le niveau et le numéro de langue comme clé d'identification artificielle.

La table lettre_motivation

L'ensemble d'entités lettre_motivation est caractérisé par les attributs suivants : le texte de la lettre et le numéro de la lettre de motivation comme clé d'identification artificielle.
Lettre_motivation_id est un code d'identification. Ce champ doit contenir un nombre unique, afin de donner une désignation unique à la rangée de la table. Ce champ est défini comme identifiant principal de cette table.
Le champ fk_candidat_id établit la relation entre le candidat et la lettre de motivation. Il s'agit d'une clé étrangère qui permet de savoir à quel candidat appartient la lettre de motivation.

La table postulation

Une postulation est constituée d'un CV, d'une lettre de motivation et d'une offre d'emploi. C'est pour cela que la table postulation contient des clés étrangères faisant référence à ces tables.

Les tables de liens

Deux tables correspondant aux liens de type complexe-complexe ont été crées : la table candidat_offre_emploi et la table recruteur_cv. En effet, un candidat peut recevoir des offres d'emploi dans son compte et un recruteur des CV.
Ces tables ont une clé primaire formée par la concaténation des clés étrangères.

6.3 Débogage et logging

Pour faciliter le débogage, la technique de *logging* a été choisie. Le principe est qu'on insère des morceaux de code de débogage afin de pouvoir afficher des informations spécifiques, comme des valeurs de variables ou des messages d'erreur.

Le projet Log4j [Log4j 2002] qui est *open source* a été choisi. Ce projet propose aux développeurs un API de *logging*. Son avantage par rapport au simple `System.out.println` est qu'il permet d'avoir plusieurs niveaux de débogage.

Pour afficher les messages, il faut d'abord créer une variable statique de type `Category`. La méthode `Category.getInstance` permet de créer une instance de `Category`. Cette méthode prend comme paramètre le nom de la classe. Dans la figure 6.4, la variable `cat` est créée et un message de type `info` affiché.

```
static Category cat =
Category.getInstance(editInscriptionAction.class.getName());
cat.info("User already logged, cannot make a new registration.");
```

Figure 6.4: exemple d'utilisation de Log4j.

Dans le fichier `WEB-INF/classes/log4j.properties`, le développeur définit la configuration. Il choisit à partir de quel niveau (`debug`, `info`, `warn` ou `error`) les messages doivent être affichés. Les messages se trouvent dans un endroit défini (*standard output* ou fichier `*.log`) La hiérarchie des classes est aussi supportée. Le fichier `log4j.properties` peut être configuré avant l'exécution (en *runtime*).

6.4 Les fichiers de configuration

6.4.1 WEB-INF/web.xml

Le fichier `web.xml` établit la configuration de l'application jobplace. Le fichier d'accueil est défini ainsi que la configuration de la servlet action, l'emplacement des librairies de balise et la restriction d'accès.

6.4.2 WEB-INF/struts-config.xml

Le fichier `struts-config.xml` est utilisé pour configurer Struts. Les form-beans, les global-forwards, les action-mappings et les actions sont enregistrés dans ce fichier.

Dans la figure 6.5, la balise `<form-beans>` définit les identifiants des beans et les types correspondants. La balise `<global-forwards>` permet définir les mots-clés des chemins pour les transferts. La balise `<action-mappings>` contient la configuration pour les différentes actions.

```
<struts-config>

  <!--  Form Bean Definitions  -->
  <form-beans>
    <form-bean name="InscriptionForm"
               type="jobplace.inscription.InscriptionForm"/>
  </form-beans>

  <!-- Global Forward Definitions  -->
  <global-forwards>
     <forward
        name="error"
        path="/pages/error.jsp"/>
  </global-forwards>

 <!--  Action Mapping Definitions  -->
  <action-mappings>
    <action path="/editInscription"
            type="jobplace.inscription.editInscriptionAction"
            name="InscriptionForm"
            scope="request"
            validate="false">
            <forward name="continue"
path="/pages/inscription/inscription.jsp"/>
    </action>
  </action-mappings>

</struts-config>
```

Figure 6.5: extrait du fichier struts-config.

6.4.3 WEB-INF/classes/poolman.xml

Le fichier `poolman.xml` permet de définir l'emplacement des bases de données. La figure 6.6 montre la configuration de la base de donnée jobplace.

```
<dbname>JOBPLACE</dbname>
<jndiName>jdbc/jobplace</jndiName>
<driver>org.gjt.mm.mysql.Driver</driver>
<url>jdbc:mysql://localhost:3306/jobplace</url>
```

Figure 6.6: la configuration de la base de donnée jobplace.

6.4.4 WEB-INF/classes/ApplicationResources.properties

Le fichier `ApplicationResources_xx.properties` contient le texte correspondant à la langue choisie pour l'affichage des données. En fait ce fichier contient des couples clé=valeur: `accueil.heading=<h1>Bienvenue &agrave; JobPlace</h1>`.

6.5 Contrôle d'accès

Le mécanisme de Tomcat est utilisé pour le contrôle d'accès. La figure 6.7 extraite du fichier `web.xml` montre les restrictions de sécurité.
La balise `<url-pattern>` décrit pour quel URI l'accès est restreint.

La balise <url-constraint> décrit les utilisateurs ayant le droit d'accès.

```
<security-constraint>
      <display-name>Example Security Constraint</display-name>
      <web-resource-collection>
         <web-resource-name>Protected Area</web-resource-name>
       <!--Define the context-relative URL(s) to be protected-->
          <url-pattern>/pages/espace-candidat/compte-
                candidat/*</url-pattern>
         <url-pattern>/do/editCV</url-pattern>
      </web-resource-collection>
      <auth-constraint>
         <!-- Anyone with one of the listed roles may access
                this area -->
         <role-name>admin</role-name>
         <role-name>candidat</role-name>
      </auth-constraint>
</security-constraint>
```

Figure 6.7: extrait de web.xml décrivant les restrictions de sécurité.

La première fois qu'un utilisateur veut accéder à une page protégée, il doit se loguer. La figure 6.8 montre l'extrait de code du fichier web.xml qui définit l'authentification.
La balise <auth-méthode> décrit le mode choisi. En général on utilise le mode BASIC qui est une boîte de dialogue. Dans le cas de jobplace, le mode FORM qui représente l'accès par un formulaire (logon.jsp) a été choisi.

```
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>Example Form-Based Authentication Area</realm-
name>
   <form-login-config>
      <form-login-page>/pages/logon.jsp</form-login-page>
      <form-error-page>/pages/error.jsp</form-error-page>
   </form-login-config>
</login-config>
```

Figure 6.8: code du mode d'authentification.

La page logon.jsp contient un formulaire ayant deux champs. Le premier champ (j_username) permet d'insérer le nom d'utilisateur et le second (j_password) le mot de passe. L'action qui est invoquée en cliquant sur le bouton Login est: response.encodeURL("j_security_check"). Cette action est définie dans Tomcat.

Pour le contrôle d'accès, le nom d'utilisateur et le mot de passe insérés sont comparés aux données enregistrées dans la bases de données users. Les tables nécessaires sont décrites dans le fichier createtablesUsers.sql (cf. Annexe A1).

6.6 Le design

L'interface utilisateur est divisée en quatre parties: le menuhaut, le menu principal, la zone principale et le menubas. Ces différentes parties sont structurées en tables.
Le menuhaut est une image représentant le logo de jobplace. Le menu principal propose des liens vers les fonctionnalités de base. Lorsque l'utilisateur clique sur un lien, les sous-menus apparaissent. Dans la zone principale, le contenu du contexte actuel de l'application est

affiché. La balise `<bean:message>` permet d'afficher le texte du fichier `ApplicationRessources_fr.properties` correspondant à la clé fixée. Par exemple `<bean:message key="accueil.heading"/>` affiche le message "Bienvenue à Jobplace". Le menubas est une image qui propose des liens vers les fonctionnalités de base.

6.7 Les paquetages

Pour chaque fonctionnalité (dépôt de CV, inscription etc.), un paquetage (*package*) a été créé. En règle générale, un paquetage est composé de deux types de classes : `Action` et `ActionForm`. Le *package* `jobplace.sql` contient les commandes et les *statements* SQL nécessaires pour l'interaction avec la base de données.
Le *package* `jobplace.common` contient les beans utilisés pour sauvegarder momentanément les données.
Le diagramme des composants de la figure 6.9 montrent les différents *packages* de l'application jobplace.

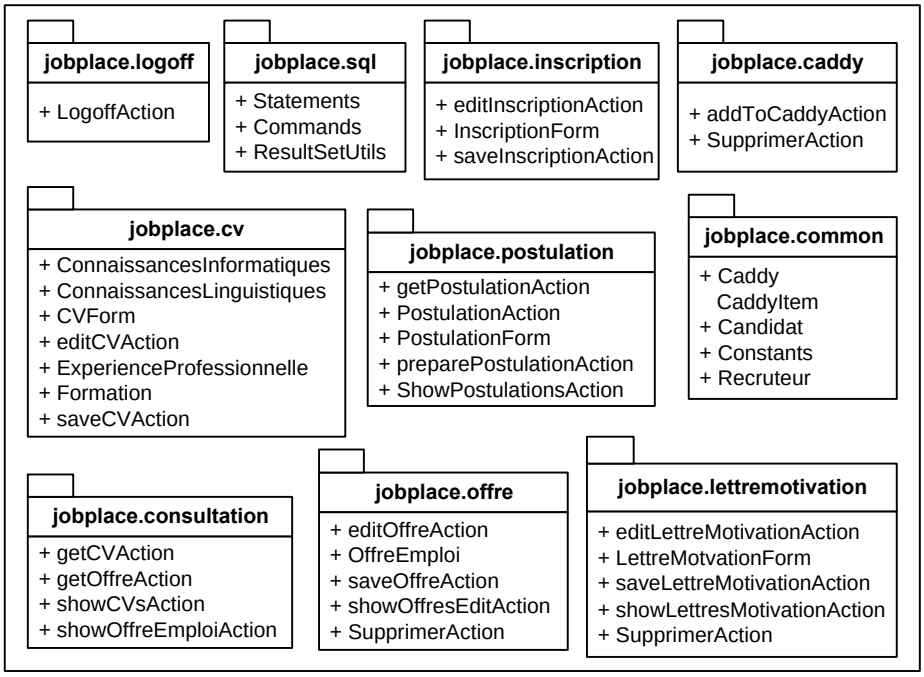


Figure 6.9: les *packages* de l'application jobplace

6.7.1 Le package jobplace.inscription

Le *package* `jobplace.inscription` sert à sauvegarder ou à modifier les données d'inscription d'un utilisateur.
Les composants du *package* `jobplace.inscription` (figure 6.10) utilisent tous la même page d'entrée, `inscription.jsp`. L'action `editInscriptionAction` est utilisée pour créer (`Create`) ou modifier (`Edit`) les informations d'inscription de l'utilisateur.
A partir des fichiers suivants : `espace-candidat.jsp`, `espace-recruteur.jsp` et `logon.jsp`, on peut accéder à la page `inscription.jsp` par l'action `/do/editInscription?action="Edit|Create"&user="candidat|recruteur".`

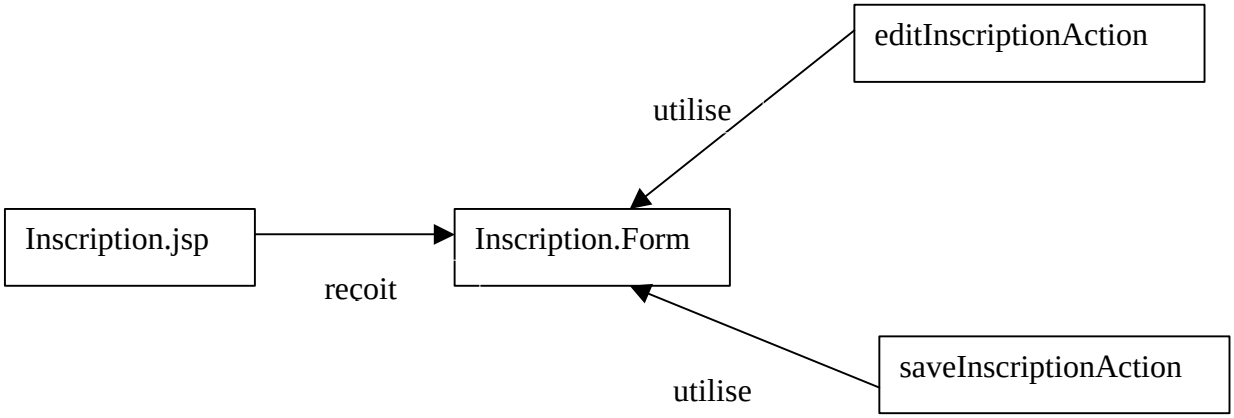


Figure 6.10: le *package* jobplace.inscription.

Les composants de ce package vont être examinés.

editInscriptionAction

La classe `editInscriptionAction` prépare le formulaire en ajoutant les données s'il en existe déjà.
L'URI `/editIncriptionAction` appelle la méthode `perform()` de la classe `editInscriptionAction`.
La variable action de la méthode `perform()` détermine si l'action est un `Create` ou un `Edit`.
Si l'action est `Create`, `editInscriptionAction` crée une instance de l'utilisateur et le sauvegarde dans le bean `InscriptionForm`.
Si l'action est `Edit`, `editIncriptionAction` contrôle s'il y a déjà un objet de type `Candidat` dans la session. Dans le cas contraire, cette action va chercher le `user_id` dans la table users qui correspond au `user_name` retourné par la méthode `request.getRemoteUser()`. Ensuite cette classe va chercher les valeurs dans la base de données et sauvegarde l'instance de l'utilisateur dans le bean `InscriptionForm`.
S'il existe un bean de l'utilisateur dans la session, `editInscriptionAction` va chercher les données de ce bean pour remplir le bean `InscriptionForm`. Cet objet utilisateur représente l'utilisateur logué. L'instance d'`InscriptionForm` permet à la cible `Inscription.jsp` de remplir ses champs.
Finalement, `editInscriptionAction` informe l'`ActionServlet` d'afficher la cible JSP qui a été définie dans le fichier `struts-config.xml`.

InscriptionForm

`InscriptionForm` contient une instance d'un bean `candidat` ou d'un bean `recruteur` et les propriétés communes (user_id, user_name, user_pass, user_role, action).

saveInscriptionAction

La classe `saveInscriptionAction` permet de sauvegarder les données.
Si l'action est égale à `Create`, le `user_role` doit être égal à `candidat` ou `recruteur` pour connaître le rôle de l'utilisateur. `saveInscriptionAction` effectue les actions suivantes:

- Sauvegarder l'utilisateur dans la base de données users (tout en contrôlant si le nom d'utilisateur existe déjà)
- Sauvegarder l'utilisateur dans la table users de la base de données jobplace.

- Sauvegarder l'utilisateur dans la table candidat ou dans la table recruteur. `saveInscriptionAction` doit d'abord chercher le `user_id` dans la table users qui sera la clé étrangère de la table candidat ou de la table recruteur.

Si l'action est égale à `Edit`, l'utilisateur existe déjà, il y a un bean utilisateur dans la session. De ce fait le `user_id` est connu, ce qui permet de sauvegarder les données.

La page inscription.jsp

La page `inscription.jsp` est utilisée aussi bien pour un nouvel utilisateur que pour un utilisateur existant.
S'il s'agit d'un utilisateur existant, `inscription.jsp` doit d'abord afficher ses informations. La balise `<logic:equal>` est utilisée pour afficher le texte adéquat concernant le type d'`action` et `user_role`. La figure 6.11 contient un exemple d'utilisation de cette balise pour l'action `Create`. L'exécution du code permet d'afficher les champs pour l'insertion du nom d'utilisateur et du mot de passe.

```
<logic:equal name="InscriptionForm" property="action"
     scope="request" value="Create">
  <tr>
    <table align="center" border="0" width="480">
      <tr>
        <td width="120">
          <bean:message key="inscription.user_name"/></td>
        <td><html:text property="user_name" size="25"
                maxlength="25"/></td>
      </tr>
      <tr>
        <td width="120">
          <bean:message key="inscription.user_pass"/></td>
        <td><html:password property="user_pass" size="25"
                maxlength="25"/></td>
      </tr>
</logic:equal>
```

Figure 6.11: exemple d'utilisation de `<logic:equal>`.

6.7.2 Le package jobplace.sql

Le package `jobplace.sql` contient trois classes: `Commands.java`, `Statements.java` et `ResultSetUtils.java`. Ces classes permettent d'interagir avec la base de données.

Commands.java

La classe `Commands.java` contient des constantes qui définissent les requêtes SQL. Un exemple de requête se trouve dans la figure 6.12.

```
/** Command to insert CV properties into CV table
*/
  public static final String CV_INSERT = "INSERT INTO "
        + CV_TABLE +
        "(cv_date,cv_objectif,fk_candidat_id) " +
        "VALUES (?,?,?)";
```

Figure 6.12: commandes pour l'insertion des données dans la table cv.

Statements.java

Toutes les méthodes de la classe `Statements.java` utilisent la démarche suivante:

- Créer un objet `Connection`.
- Préparer un `Statement`.
- Fixer les valeurs.
- Fermer la connexion.

Après avoir obtenu un objet `Connection`, il est possible de créer les instructions SQL. L'objet `Connection` peut renvoyer des objets `Statement` ou `PrepareStatement`.

Pour qu'un objet `Statement` accomplisse une action quelconque, il faut lui fournir du code SQL à exécuter. Pour cela, une requête SQL est transmise comme paramètre `String` (ces requêtes se trouvent dans la classe `Commands.java`). Par la suite, la méthode d'exécution `executeQuery()` est invoquée. Cette méthode renvoie un objet `ResultSet` renfermant le résultat de la requête SQL. La méthode `executeUpdate` est utilisée pour faire une actualisation, une insertion ou une suppression des données. Sa valeur de retour est positive si la requête a abouti.

Il y a cinq types de méthode dans la classe `Statements`: `updateXXX()`, `queryXXX()`, `execQueryBean()`, `execQueryXXX()` et `deleteQuery()`.

La méthode `updateXXX()` permet d'actualiser ou d'insérer des données dans la table. Elle reçoit les données à actualiser comme paramètre.

La méthode `queryXXX()` permet de chercher les valeurs dans la base de données et remplir un array avec ces données. Il faut utiliser pour cela la méthode `ResultSetUtils.populateArray()` qui retourne un *array* de bean.

La méthode `execQueryBean()` permet de chercher les valeurs dans la base de données et de remplir un bean avec ces données. Il faut utiliser pour cela la méthode `ResultSetUtils.populate()`.

La méthode `execQueryXXX()`permet d'exécuter une requête. Elle permet par exemple de chercher l'identifiant d'un utilisateur.

La méthode `deleteQuery()` permet de supprimer les données à l'aide de l'identifiant.

ResultSetUtils.java

Les méthodes de la classe `ResultSetUtils.java` permettent de remplir les données de l'objet `ResultSet` – obtenue par l'exécution d'un `Statement` – dans un bean.
Une méthode de la classe `ResultSetUtils` remplit un objet de type `Hashmap` par des pairs (clé-valeur) qui ont comme clé le nom du champ et comme valeur, la valeur correspondante. Pour chaque champ du bean, la méthode vérifie s'il y a une clé correspondant à ce champ. Dans ce cas, la méthode remplit le bean avec la valeur correspondante.

6.7.3 Le package jobplace.offre

Ce package sert à créer de nouvelles offres d'emploi, à les modifier ou supprimer. Les chemins pour y accéder se trouvent dans le compte recruteur.

La figure 6.13 montre le premier groupe de composants de ce package. Ces composants seront examinés par la suite.

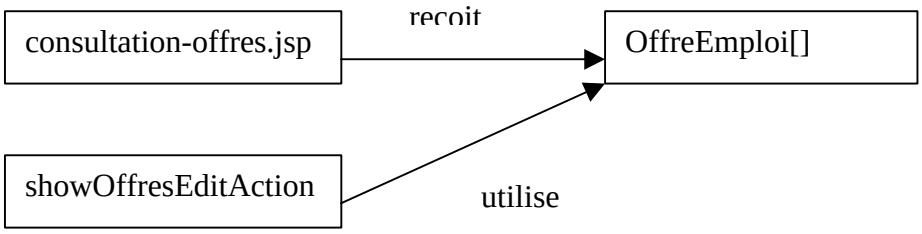


Figure 6.13: les composants pour la consultation des offres d'emploi.

showOffresEditAction

La classe `showOffresEditAction` va chercher toutes les offres d'emploi du recruteur dans la base de données et les enregistre dans un *array* de bean `OffreEmploi`. Ensuite, elle les affiche à l'aide de `consultation-offres.jsp`. Trois possibilités se présentent :

- La création d'une nouvelle offre d'emploi: la méthode `editOffreAction` avec le paramètre `Create` est appelée.
- L'utilisateur clique sur un lien pour modifier une offre d'emploi, dans ce cas, la méthode `editOffreAction` est appelée avec le paramètre `Edit`.
- L'utilisateur clique sur un lien pour supprimer une offre d'emploi, dans ce cas, la méthode `supprimerOffreAction` est appelée

La figure 6.14 montre des extraits de code des fichiers `consultation-offres.jsp` et `OffreEmploi.java`. Ce code permet de passer un paramètre à la requête qui est créée en cliquant sur le lien. Dans l'exemple l'URI `editOffre?action=Edit& offre_emploi_id=3` est appelée. Le paramètre `offre_emploi_id` et la valeur correspondante sont contenus dans la propriété `mapping`. Le `mapping` est défini par la méthode `setMapping()` de l'`OffreEmploi`.

```
<html:link page="/do/editOffre?action=Edit" name="offreemploi"
     property="mapping">
  <bean:write name="offreemploi" property="offre_emploi_poste"/>
</html:link>

public void setMapping() {
  mapping.put(Constants.OFFRE_EMPLOI_ID, new
     Integer(offre_emploi_id));
}
```

Figure 6.14: extrait des fichiers `consultation-offres.jsp` et `OffreEmploi.java`.

La figure 6.15 montre le deuxième groupe de composants du *package* `jobplace.offre`. Ces composants seront examinés par la suite.

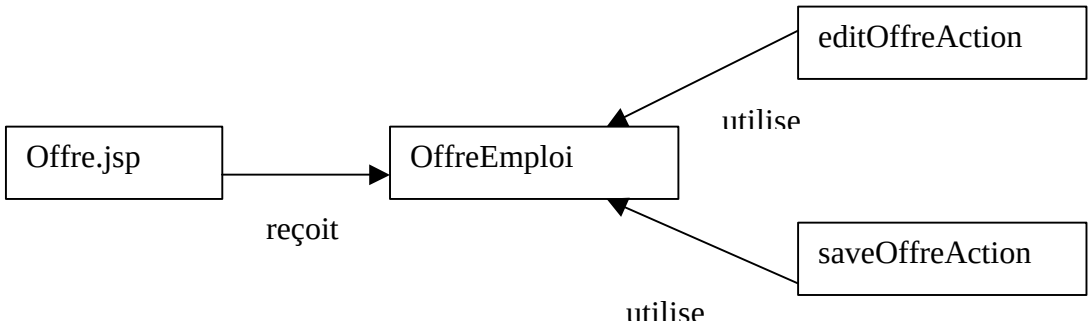


Figure 6.15: les composants pour la création d'une offre d'emploi.

editOffreAction

La classe `editOffreAction` prépare le formulaire qui se trouve dans la page `Offre.jsp` en ajoutant les données s'il en existe déjà.
L'URI `/editOffre?action=Create` appelle la méthode `perform()` de la classe `editOffreAction` pour créer une nouvelle offre d'emploi. Elle crée une nouvelle instance du bean `OffreEmploi` et la sauvegarde dans la requête. Par la suite le formulaire de la page `Offre.jsp` reçoit la requête.
Pour modifier une offre d'emploi on clique sur le lien correspondant. L'URI `/editOffre?action=Edit&offre_emploi_id=2` est invoqué. Dans ce cas il existe un *array* `OffreEmploi` dans la session. `EditOffreAction` cherche dans l'*array* le bean qui correspond à `l'offre_emploi_id` (cf. figure 6.16) et le sauvegarde dans la requête. Ceci permet à la cible `Offre.jsp` de remplir ses champs avec les valeurs contenues dans le bean `OffreEmploi`.
Finalement, L'`editOffreAction` informe l'`ActionServlet` d'afficher la cible JSP qui a été définie dans le fichier `struts-config.xml`: `<forward name="continue" path="/pages/espace-recruteur/compte-recruteur/Offre.jsp"/>`

```
// get offre_emploi corresponding to offre_emploi_id
for (int i=0;i<offres.length;i++) {
     if (offre_emploi_id == offres[i].getOffre_emploi_id()) {
                offre_emploi = offres[i];
                break;
     }
}
```

Figure 6.16 : code pour chercher une offre d'emploi dans l'*array*.

saveOffreAction

`SaveOffreEmploiAction` permet de sauvegarder les données dans la base de données.
Si le paramètre action est égal à `Create` la méthode `Statements.updateOffre()` est utilisée avec la requête SQL `Commands.OFFRE_INSERT`. Si l'action est égale à `Edit`, c'est la requête `Commands.OFFRE_UPDATE` qui est utilisée.

6.7.4 Le package jobplace.lettremotivation

La classe `showLettresMotivationAction` va chercher toutes les lettres de motivation dans la base de données et les enregistre dans un *array* de bean `LettreMotivationForm`. Ensuite, elle les affiche à l'aide de `selectLettreMotivation.jsp`. Trois possibilités se présentent :

- La création d'une nouvelle lettre de motivation: la méthode `editLettreMotivationAction` avec le paramètre `Create` est appelée.
- L'utilisateur clique sur un lien pour modifier une lettre de motivation, dans ce cas, la méthode `editLettreMotivationAction` est appelée avec pour paramètre `Edit`.
- L'utilisateur clique sur un lien pour supprimer une lettre de motivation, dans ce cas, la méthode `supprimerLettreMotivationAction` est appelée.

editLettreMotivationAction

La classe `editLettreMotivationAction` prépare le formulaire en ajoutant les données s'il en existe déjà.
Dans un premier temps, l'URI `/editLettreMotivationAction` appelle la méthode `perform()` de la classe `editLettreMotivationAction`. La méthode `perform()` établit les variables session et action.
L'URI `/editLettreMotivation?action=Create` appelle la méthode `perform()` de la classe `editLettreMotivationAction` pour créer une nouvelle lettre de motivation. Elle crée une nouvelle instance du bean `LettreMotivationForm` et la sauvegarde dans la requête. Par la suite le formulaire de la page `LettreMotivation.jsp` reçoit la requête.
Pour modifier une lettre de motivation on clique sur le lien correspondant. L'URI `/editLettreMotivation?action=Edit&lettre_motivation_id=1` est invoqué. Dans ce cas il existe un *array* `LettreMotivationForm` dans la session. `EditLettreMotivationAction` cherche dans l'*array* le bean qui correspond à la `lettre_motivation_id` et le sauvegarde dans la requête. Ceci permet à la cible `LettreMotivation.jsp` de remplir ses champs avec les valeurs contenues dans le bean `OffreEmploi`.

saveLettreMotivationAction

`SaveLettreMotivationAction` permet de sauvegarder les données dans la base de données. Si le paramètre action est égal à `Create` la méthode `Statements.updateLettreMotivation()` est utilisée avec la requête SQL `Commands.LETTRE_MOTIVATION_INSERT`. Si l'action est égale à `Edit`, c'est la requête `Commands.LETTRE_MOTIVATION_UPDATE` qui est utilisée.

6.7.5 Le package jobplace.cv

Le package `jobplace.cv` sert à créer un nouveau CV ou à modifier un CV existant. Les composants de ce package sont montrés dans la figure 6.17.

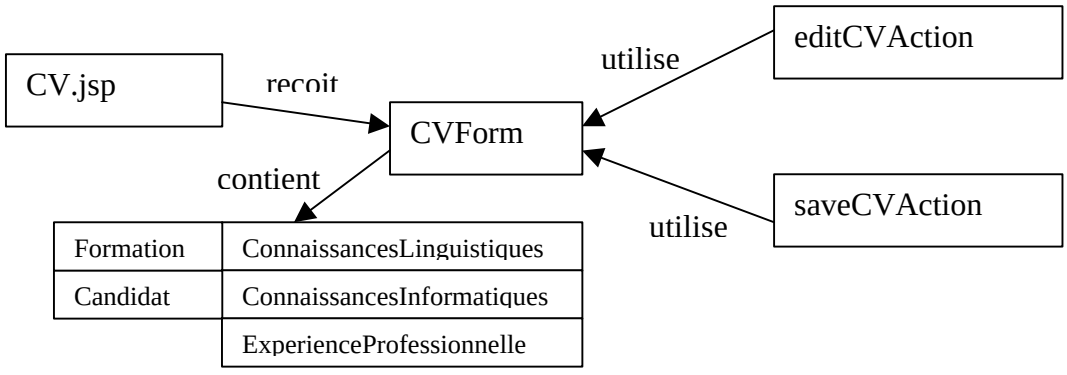


Figure 6.17: composant du *package* `jobplace.cv`.

Les composants du *package* `jobplace.cv` utilisent tous la même page d'entrée, `CV.jsp`. Les classes `editCVAction` et `saveCVAction` fonctionne comme les classes correspondantes des *package* `jobplace.lettremotivation` et `jobplace.offre`.

La table cv a des liens avec d'autres tables. Leurs données sont sauvegardées dans les beans correspondants (`ConnaissancesLinguistiques`, `ConnaissancesInformatiques`, `ExperienceProfessionnelle`, `Formation` et `Candidat`). `CVForm` contient une instance de chacun.

6.7.6 Le package jobplace.logoff

Le *package* `jobplace.logoff` contient la classe `LogoffAction` qui permet d'annuler la session.

6.7.7 Le package jobplace.common

Le *package* `jobplace.common` contient des classes communes à tous les *package* de l'application jobplace. Ce sont les classes `Candidat`, `Recruteur`, `Caddy`, `CaddyItems` et `Constants`.
Les classes `Candidat` et `Recruteur` sont utilisées pour enregistrer les données de l'utilisateur. Ce sont des beans avec des méthodes `setXXX()` et `getXXX()`.
La classe `Caddy` permet de collecter les objets de type `CaddyItems` qui est une interface. `Jobplace.cv.CVForm` et `jobplace.offre.OffreEmploi` implémentent cette interface, ce sont donc ces objets qui peuvent être collectés avec le caddie. La figure 6.18 montre la méthode `addCaddyItem` qui ajoute l'objet `caddyItem` dans le caddie. `CaddyItems` est un objet de type `Hashmap` qui représente le caddie. L'id de l'article est transformé en objet de type `Integer` et représente la clé du couple dans le `Hashmap`.

```
public void addCaddyItem(CaddyItem caddyItem) {
      caddyItems.put(new Integer(caddyItem.getId()), caddyItem);
      setItemCount();
}
```

Figure 6.18: la méthode `addCaddyItem`.

La classe `Constants` contient des constantes qui sont les clés pour les attributs de session et les paramètres de requête:
`public static final String CANDIDAT_KEY = "candidat_key";`
La ligne de code `Constants.CANDIDAT_KEY` retourne la valeur correspondante: "candidat_key".

6.7.8 Le package jobplace.caddy

Le package `jobplace.caddy` sert à ajouter ou supprimer un article du caddie.
A partir de `consultation-cvs.jsp`, `showCV.jsp`, `showOffres.jsp` ou de `consultation-offres.jsp`, les actions `addToCaddyAction` ou `SupprimerAction` peuvent être appelées. Après l'exécution de la requête, l'utilisateur se retrouve à la page de départ.
La classe `addToCaddyAction` reconnaît la page de départ à l'aide du paramètre `input-path`. L'`input-path` permet également de savoir s'il s'agit d'ajouter une offre ou un CV dans le caddie. La figure 6.19 montre les différentes valeurs du paramètre `input-path`. L'attribut `name` équivaut `input-path`. L'attribut `path` définit le chemin de la page à afficher.

```
<action path="/addToCaddy"
      type="jobplace.caddy.addToCaddyAction">
   <forward name="consultation-offres" path="/pages/espace-
          candidat/consultation-offres.jsp"/>
   <forward name="showOffre" path="/pages/espace-
          candidat/showOffre.jsp"/>
   <forward name="consultation-cvs" path="/pages/espace-
          recruteur/consultation-cvs.jsp"/>
   <forward name="showCV" path="/pages/espace-
          recruteur/showCV.jsp"/>
</action>
```

Figure 6.19: extrait du fichier `struts-config.xml`.

Dans un premier temps, une instance de la classe `Caddy` est créée, la propriété `kind` est posée sur "cv" ou "offre". Ensuite, le bean qui correspond à l'id (paramètre de la requête) est cherchée dans l'*array* (cf. figure 6.16). Le bean est ajouté au caddie par la méthode `addCaddyItem` (cf. figure 6.18).

La classe `SupprimerAction` cherche l'article qui correspond à l'id (paramètre de la requête):
`CaddyItem item = caddy.getCaddyItem(id);`
et l'efface: `caddy.deleteCaddyItem(item);`

6.7.9 Le package jobplace.consultation

Ce package sert à consulter les offres d'emploi ou les CV.
La classe `showOffresEmploiAction` va chercher toutes les offres d'emploi des recruteurs dans la base de données et les enregistre dans un *array* de bean `OffreEmploi`. Cet *array* est mise dans la session. Ensuite, la classe appelle la page `consultation-offres.jsp` qui affiche les offres d'emploi à l'aide de la balise `<logic:iterate>`. Cette balise permet de faire la même opération pour chaque objet de l'*array*.
Dans la figure 6.20, il y a un *array* `offres_emploi` qui contient des objets de type `jobplace.offre.OffreEmploi`. Le nom temporaire de l'objet est `offreemploi` (`id`). Pour chaque objet la valeur du champs `offre_emploi_date` est affiché à l'aide de la balise `<bean:write>`.

```
<logic:iterate id="offreemploi"
      type="jobplace.offre.OffreEmploi"
      name="<%=Constants.OFFRES_EMPLOI_ARRAY_KEY%>">
   <bean:write name="offreemploi" property="offre_emploi_date"/>
</logic:iterate>
```

Figure 6.20: la balise `<logic:iterate>`.

La classe `getOffreAction` prend la valeur du paramètre `offre_emploi_id` de la requête. Elle cherche dans l'*array* le bean `OffreEmploi` qui correspond à cet id et envoie le contrôle à la page `showOffre.jsp`. La page affiche les données du bean choisi (cf. figure 6.21).

```
<bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
      property="offre_emploi_id"/>
```

Figure 6.21: code qui affiche la valeur de `offre_emploi_id` du bean.

La classe `showCVsAction` va chercher tous les CV des candidats dans la base de données et les enregistre dans un *array* de bean CV. Ensuite, elle les affiche à l'aide de `consultation-cvs.jsp`.

La classe `getCVAction` va chercher un CV correspondant à `cv_id` dans l'*array* et l'affiche à l'aide du fichier `showCV.jsp`.

6.7.10 Le package jobplace.postulation

Ce package contient des classes permettant de postuler ou de consulter les postulations.
Un candidat peut postuler à partir des fichiers `consultations-offres.jsp`, `showOffre.jsp` ou `caddy.jsp`. En cliquant sur le lien correspondant, la classe `preparePostulationAction` est invoquée.

preparePostulationAction

Cette classe réunit les composants nécessaires pour postuler: l'offre d'emploi, le CV et la lettre de motivation. Elle contrôle si ces composants existent déjà. Si tel n'est pas le cas, elle demande à l'utilisateur de les créer.
Tous les composants sont enregistrés dans `PostulationForm`. L'objet `PostulationForm` est mis dans la session afin de pouvoir y accéder pendant toutes les étapes. `PostulationForm` contient un champ "action" qui décrit l'étape de la préparation. On distingue les étapes de préparation suivantes:

- `Step-Candidat` – Cette étape crée une instance du candidat.
- `Step-Offre` – Cette étape contrôle l'existence d'une offre pour la postulation.
- `Step-CreateCV` – Cette étape crée une instance d'un `CVForm`. Si l'utilisateur n'a pas de CV, la classe `editCVAction` est invoquée.
- `Step-CreateLettreMotivation` – Cette étape consiste en la création d'une lettre de motivation: la classe `editLettreMotivationAction` est appelée.
- `Step-LettreMotivation` – Cette étape cherche toutes les lettres de motivation dans la base de données et les sauvegarde dans un *array*.
- `Step-Postulation` – Cette étape représente la phase finale. Le contrôle est envoyé à la page `postulation.jsp`.

postulation.jsp

La page `postulation.jsp` affiche les composants de la postulation suivants:

- `Offre_emploi_poste` et `offre_emploi_id` pour l'offre d'emploi;
- `Cv_objectif` pour le CV;
- Un menu déroulant pour la sélection de la lettre de motivation.

La figure 6.22 montre le code du menu déroulant pour la sélection de la lettre de motivation. La balise `<html:select>` définit le menu déroulant. L'attribut `property` décrit dans quel champ la valeur choisit doit être enregistrée.
La valeur de retour de la balise `<html:option>` est l'id de la lettre sélectionnée.

```
<html:select property="lettreform.lettre_motivation_id">
     <logic:iterate id="lettremotivation" type="jobplace.
          lettremotivation.LettreMotivationForm"
          name="<%= Constants.LETTRE_MOTIVATION_ARRAY_KEY %>">
          <html:option value="<%=Integer.toString
               (lettremotivation.getLettre_motivation_id())>">
               <bean:write name="lettremotivation"
                    property="lettre_motivation_texte"/>
          </html:option>
     </logic:iterate>
</html:select>
```

Figure 6.22: code du menu déroulant pour la sélection de la lettre de motivation.

PostulationAction

La classe `PostulationAction` exécute la postulation en insérant les clés étrangères des tables offre_emploi, lettre_motivation, cv et la date de la postulation dans la table postulation.

ShowPostulationAction

La classe `ShowPostulationAction` va chercher toutes les postulations des candidats dans la base de données et les enregistre dans un *array* de bean `PostulationForm`. Ensuite, elle les affiche à l'aide de `consultation-postulations.jsp`.

getPostulationAction

La classe `getPostulationAction` prend la valeur du paramètre `postulation_id` de la requête. Elle cherche dans l'*array* le bean `PostulationForm` qui correspond à cet id et envoie le contrôle à la page `showPostulation.jsp`. La page affiche les données du bean `PostulationForm`.

7 Mode d'emploi

7.1 Les instructions d'installation

Pour assurer le fonctionnement de l'application les fichiers ci-dessus sont requis:

- Fichier compressé Tomcat 4.0.1: `jakarta-tomcat-4.0.1.zip`;
- exécutable d'installation JDK1.3: `j2sdk1_3_0-win.exe`;
- fichier compressé MySQL: `mysql-3_23_49-win.zip`;
- fichier compressé de l'application jobplace: `jobplace.war`;
- le fichier du driver MySQL: `mm.mysql-2.0.4-bin.jar`;
- le répertoire `data` qui contient les fichiers `*.sql` pour la création des tables;
- le fichier `my.cnf` pour la configuration de MySQL.

Tous les fichiers et programmes se trouvent sur le cd en annexe E et/ou sont téléchargeables depuis les sites suivants:

- [Apache 2002]
- [Sun Microsystems 2002]
- [MySQL 2002]

Il convient de remarquer les points suivants :

- Les informations contenues entre crochets doivent être remplacées par les valeurs réelles (par exemple: [chemin vers Tomcat] devient `c:\serveur\tomcat`)
- Eviter de mettre des espaces dans les noms des répertoires contenant les différentes programmes.
- Si le nom des répertoires dépassent 8 caractères et contiennent des espaces, prendre les 6 premiers caractères et ajouter ~1 (par exemple `c:\program files\jdk` devient `c:\progra~1\jdk`).

7.1.1 Installation de MySQL

- Décompresser `mysql-3_23_49-win.zip` dans un répertoire.
- Lancer l'exécutable setup.exe et suivre les instructions.
- Ajouter le fichier `my.cnf` dans le répertoire c:\.
- Exécuter depuis une fenêtre DOS la commande suivante:

    - Sous Windows 95/98
      ```
      ->[MySQL]/bin/mysqld
      ```
    - Sous Windows NT/2000
      ```
      ->[MySQL]/bin/mysqld-nt --install
      ->net start mysql
      ```

7.1.2 Création des bases de données

Exécuter depuis une fenêtre DOS les commandes suivantes:

```
->cd [data]
->set path=%path%;[MySQL]\bin
->mysql
mysql>create database users;
mysql>use users;
```

```
mysql>source createtablesUsers.sql;
mysql>create database jobplace;
mysql>use jobplace;
mysql>source createtablesJobplace.sql;
```

7.1.3 Installation de Tomcat 4.0.1

- Décompresser Tomcat dans un répertoire.
- Editer le fichier `[Tomcat]\bin\catalina.bat` et ajouter au début du fichier les lignes:
  ```
  set JAVA_HOME=[chemin vers JDK]
  set CATALINA_HOME=[chemin vers Tomcat]
  ```
- Editer le fichier `[Tomcat]\conf\server.xml` en mettant en commentaire les lignes:
  ```
  <Realm className="org.apache.catalina.realm.MemoryRealm" />
  ```
  Ajouter les lignes:
  ```
  <Realm className="org.apache.catalina.realm.JDBCRealm"
        debug="99" driverName="org.gjt.mm.mysql.Driver"
  connectionURL="jdbc:mysql://localhost:3306/users"
       userTable="users" userNameCol="user_name"
       userCredCol="user_pass" userRoleTable="user_roles"
       roleNameCol="role_name"/>
  ```
- Ajouter le fichier `mm.mysql-2.0.4-bin.jar` dans le répertoire `[Tomcat]\common\lib`.
- Ajouter le dossier `jobplace.war` dans le répertoire `[Tomcat]\webapps`.
- Exécuter depuis une fenêtre DOS la commande `[Tomcat]\bin\startup`.
- Ouvrir un navigateur et tester l'adresse `http://localhost:8080`.

7.2 Utilisation de l'application Web

Pour l'utilisation de l'application sur un serveur local, il convient d'utiliser l'adresse `http://localhost:8080/jobplace`. La page d'accueil de l'application jobplace va s'afficher (cf. figure 7.1).

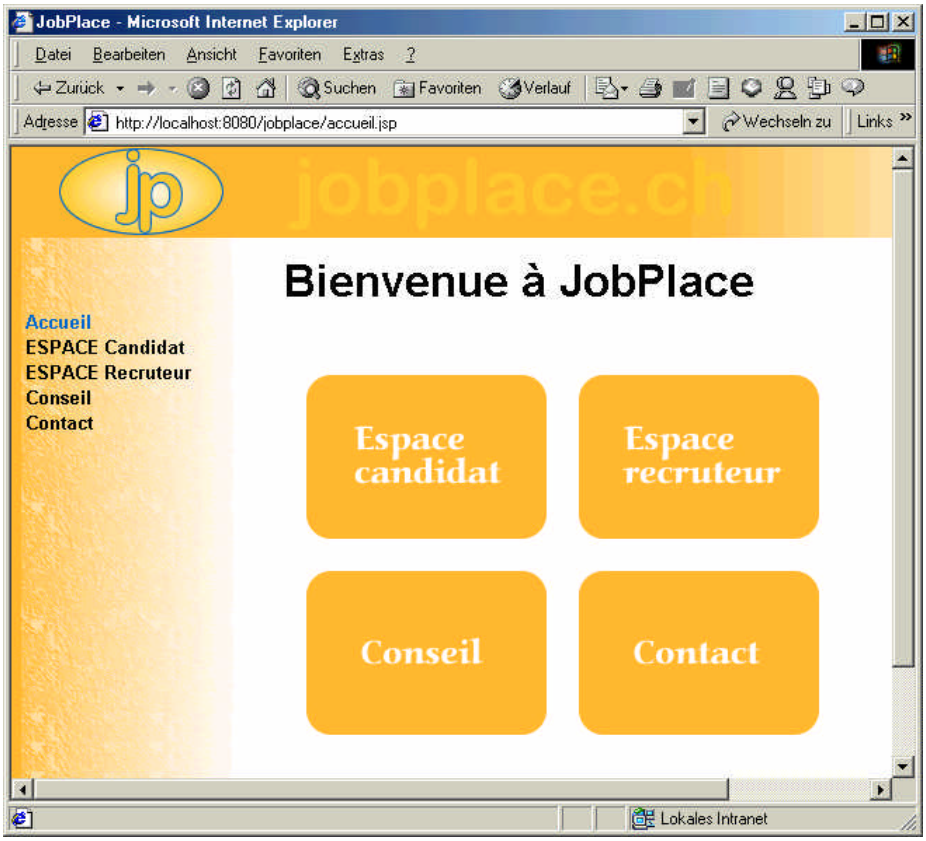


Figure 7.1: page d'accueil de l'application jobplace.

La page d'accueil de l'application contient le menu principal du site Web. Des liens hypertextes donnent la possibilité à l'utilisateur d'accéder directement aux rubriques qui l'intéressent. Le menu principale est composé des pages Accueil, ESPACE Candidat, ESPACE Recruteur, Conseil et Contact. Ces pages constituent également des menus qui peuvent être subdivisés en sous-menus.

7.2.1 ESPACE Candidat

Si l'utilisateur clique sur le lien ESPACE Candidat, il dispose des services suivants: inscription, consultation des offres d'emploi, postuler ou accéder au compte.
Pour s'inscrire, l'utilisateur clique sur le lien `Inscription`. Il remplit un formulaire HTML et soumet l'information au serveur.
Pour consulter les offres d'emploi, l'utilisateur clique sur le lien `Consultation des offres`. En cliquant sur une offre choisie, le détail s'affiche. Le candidat peut collecter les offres d'emploi dans un caddie. Il a la possibilité de postuler directement après la consultation d'une offre, ou alors à partir du caddie. Seuls les candidats inscrits peuvent postuler de manière interactive. Pour pouvoir postuler, il faut joindre à l'offre d'emploi un CV et une lettre de motivation disponibles sur le compte personnel. Si ces documents n'existent pas encore, ils peuvent être créés.
Pour accéder à son compte, l'utilisateur doit fournir à l'application une combinaison valide de *password* (mot de passe) et de *username* (nom d'utilisateur). Le compte candidat (figure 7.2) permet d'enregistrer le CV et les lettres de motivation ou de modifier les données (profil, CV et lettres de motivation).

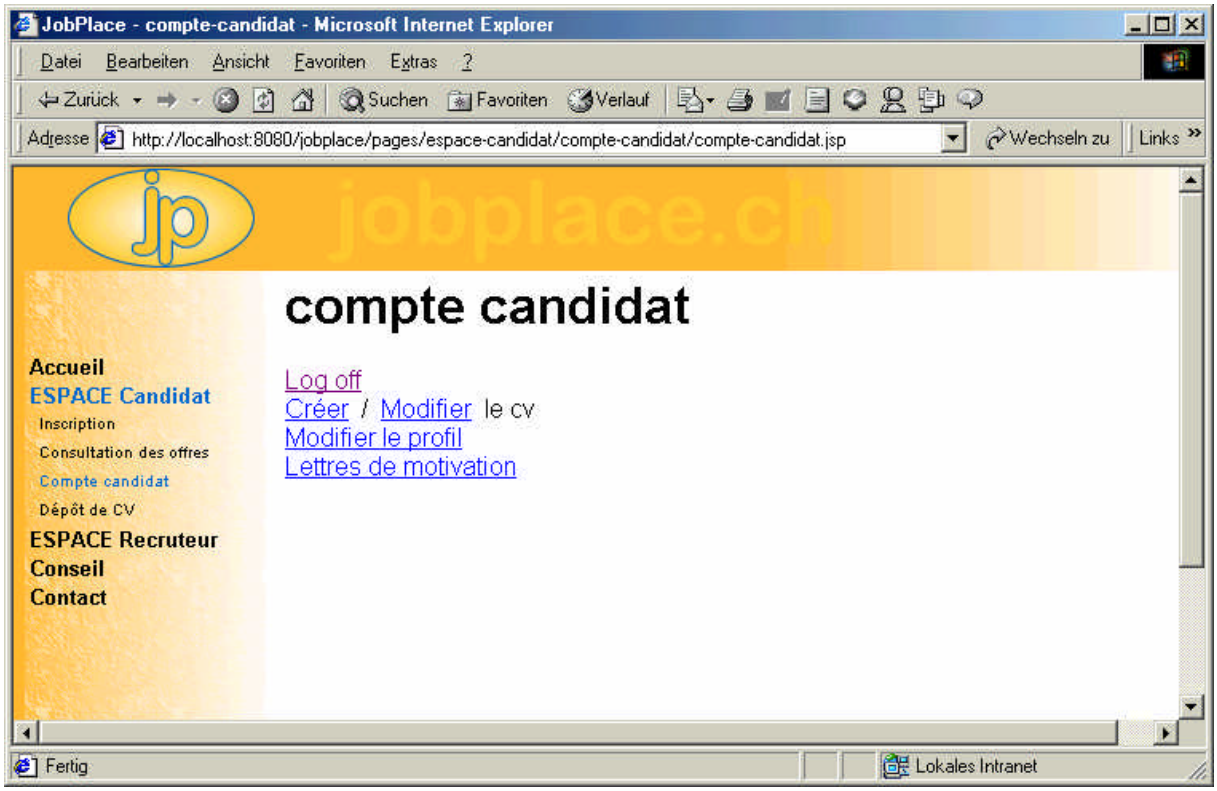


Figure 7.2: Compte Candidat

7.2.2 ESPACE Recruteur

Si l'utilisateur clique sur le lien ESPACE Recruteur, il a le choix entre s'inscrire, consulter la cvthèque ou accéder à son compte.
Pour s'incrire, le recruteur clique sur le lien `Inscription`. Il peut ainsi remplir un formulaire.
Pour consulter les CV, le recruteur clique sur le lien `Consultation des cv`. En cliquant sur un CV choisi, le détail s'affiche. Le recruteur a également la possibilité de collecter les CV dans un caddie. Il a la possibilité d'envoyer un CV sélectionné ou les CV du caddie dans son compte pour le traitement.
L'accès au compte recruteur est validé par un nom d'utilisateur et un mot de passe. Le compte recruteur permet de modifier le profil utilisateur, de gérer les offres d'emploi et consulter les postulations (figure 7.3).

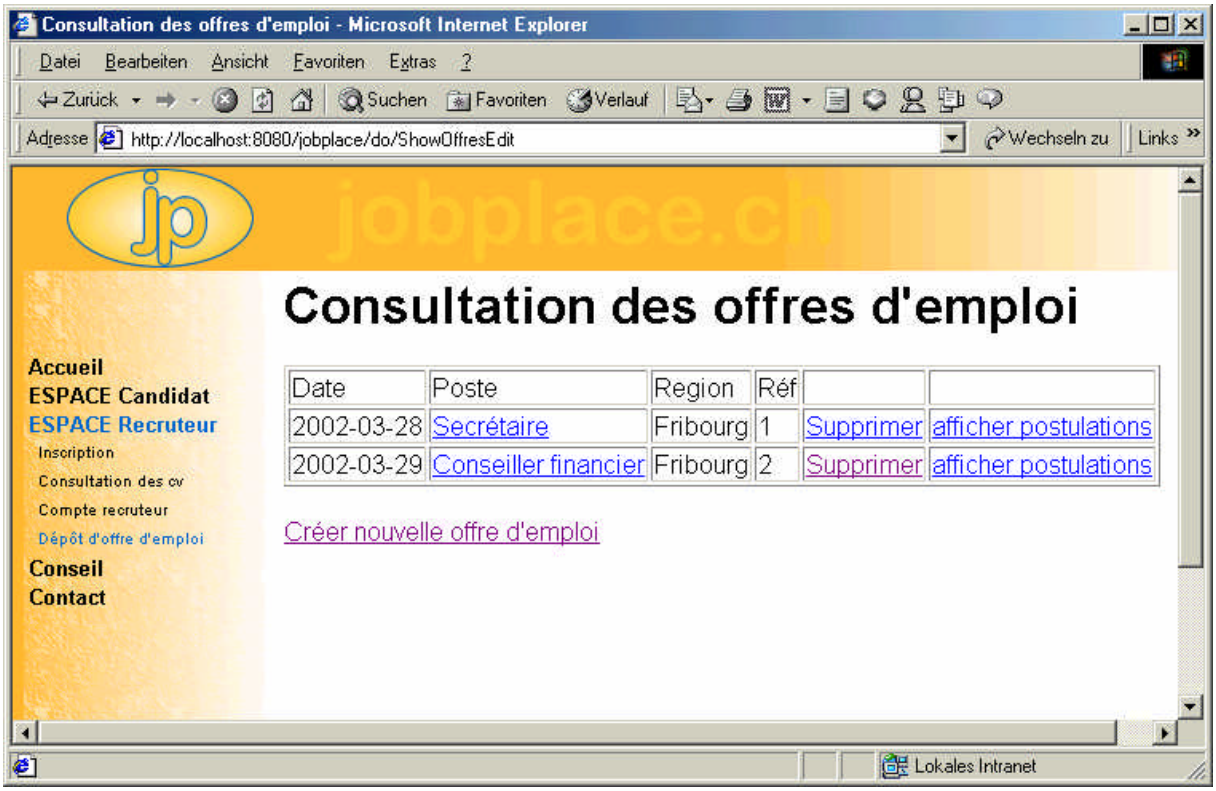


Figure 7.3: la consultation des offres d'emploi permet d'afficher les postulations.

8 Conclusion

L'expansion du Web dans le secteur de l'emploi a donné naissance au développement de nouveaux outils: les sites d'emplois. L'utilisation des sites d'emplois apporte des solutions nouvelles aux recruteurs comme aux candidats. Les premiers peuvent aujourd'hui recevoir des candidatures sans avoir à ouvrir les enveloppes. Par la suite, l'envoi des réponses, et le dialogue avec le candidat s'effectuent aussi électroniquement. Les candidats ont, quant à eux, la possibilité de chercher un emploi quand et où ils le désirent.

L'objectif de ce travail était de montrer, à travers un exemple comment les technologies Internet peuvent être utilisées, pour implémenter un espace d'échanges interactif afin d'optimiser le processus de recrutement.
L'optimisation du processus de e-recrutement implique la connaissance des acteurs de l'emploi sur le Web.

Il existe plusieurs acteurs de l'emploi sur Internet: les moteurs de recherche, les portails, les forums de discussion, les pages d'entreprises, les sites de média, les agences de placement de personnel et les marchés virtuels d'emplois.

Cependant, construire un bon site d'emplois implique de comprendre certains aspects en constante évolution et d'intégrer des technologies Internet.
C'est dans cet ordre d'idées que ce travail développe une solution complète de e-recrutement où les technologies et langages les plus récents sont utilisés: UML, Java, JSP, Struts, et SQL.
Quelles sont donc les aspects importants de la création d'un site d'emplois et comment développer un catalogue d'emplois en ligne?

Avant de développer une application, il faut d'abord analyser les exigences du sytème. C'est la première étape du développement d'un projet. La détermination et la compréhension des besoins sont souvent difficiles à cause de la quantité d'informations disponible. Or, comment mener à bien un projet si l'on ne pas où l'on va? Il faut donc modéliser les besoins du système. Les diagrammes de cas d'utilisation du langage de modélisation UML permettent de modéliser les besoins des utilisateurs d'un système. Une fois identifiés et structurés, ces besoins permettent d'identifier les fonctionnalités principales du système.

La seconde étape du processus de développement consiste à définir l'architecture. Pour cela il faut faire le choix des technologies. JSP (JavaServer Pages) est une technologie Java permettant de créer des sites Web dynamiques accédant à des bases de données. L'avantage de JSP par rapport aux autres langages de script côté serveur est sa portabilité sur de nombreuses plates-formes (Windows, Unix, Linux, etc.).
L'utilisation d'un *framework* comme Struts demande au début l'apprentissage de nouvelles balises. Mais avec la séparation claire de la présentation de la logique, le développement d'une application Web complexe est facilité.

La plupart des applications dépendent d'un mécanisme de sauvegarde d'arrière-plan – la base de données – avec lequel elles doivent interagir. Les informations de cette base de données peuvent être sauvegardées, recherchées, interrogées ou extraites. Le SQL est le langage d'interaction avec les bases de données le plus populaire.

La dernière étape du processus de développement concerne l'implémentation de l'application en fonction des technologies choisies. Cette étape a abouti à la mise en place d'un système de recrutement virtuel.

Ce travail est un outil qui donne au lecteur un mode d'emploi permettant de passer de la réflexion à la pratique. Ainsi dans chaque chapitre se trouvent des exemples pour la mise en œuvre.
Cependant, pour perfectionner cet outil, les points forts et faibles seront analysés et les possibilités d'améliorations proposées.

Points forts

- Ce travail met à la disposition du lecteur une bibliothèque de sites d'emplois.
- La description des scénarios montre le fonctionnement idéal d'un web-shop d'emplois.
- L'implémentation de l'application est simple et permet des développements ultérieurs.
- La séparation de la logique et de la présentation permet à un utilisateur avec peu de connaissances en informatique de changer facilement l'interface graphique de l'application Web.
- L'utilisation des caractéristiques d'internationalisation Java permet d'implémenter plusieurs langues.
- L'application est destinée à être indépendante de la plate-forme.
- Tous les composants utilisés pour le développement de l'application sont non-propriétaires.

Points faibles

- Le logiciel MySQL n'est pas très convivial. Par exemple *query-by-example* n'est pas supporté.
- L'utilisation du mécanisme de contrôle d'accès de Tomcat rend l'application Web dépendante du serveur.
- L'utilisation de COMMIT et ROLLBACK pour les transactions nécessite plusieurs modifications dans le code.

Extensions possibles

- Ajouter d'autres langues à l'interface utilisateur. Il suffit de créer un fichier `ApplicationResources_<mot clé de la langue>.properties`. Ensuite, il faut changer l'image du fichier `menubas-<mot clé de langue>.gif` pour adapter le texte de l'image à la langue. Il en est ainsi pour les images `conseil.gif`, `contact.gif`, `espace- recruteur.gif` et `espace-candidat.gif`.
- Implémenter les pages conseil et contact.
- Pouvoir insérer plusieurs types de formations, d'expériences professionnelles, de connaissances linguistiques et de connaissances informatiques dans le CV.
- Améliorer la fonction de consultation:
    - Implémenter une recherche par critère qui traquerait les offres et les CV correspondant aux critères indiqués par le candidat, respectivement par le recruteur. Pour cela, il suffit de créer une page JSP qui propose des menus déroulants pour le choix des critères. En outre, les requêtes SQL doivent être spécifiées.

    - Structurer l'affichage des offres d'emploi et des CV (par secteur d'activité, par ordre chronologique de parution etc.).
- Etendre la fonction de gestion des postulations chez le recruteur. Il pourra ainsi traiter les candidatures dans son compte (convoquer les candidats, garder sous la main les CV les plus intéressants, mettre en attente les CV moins ciblés, écarter les CV qui ne sont pas intéressants).
- Permettre aux utilisateurs de changer le nom d'utilisateur et le mot de passe. Il faudrait implémenter un mécanisme pour l'actualisation de ces champs dans les tables users des bases de données users et jobplace.
- La gestion des transactions pourrait être implémentée à l'aide de COMMIT et ROLLBACK. Il faudrait changer l'implémentation de telle manière qu'une connexion ne soit plus utilisée pour une seule commande SQL mais pour toutes les commandes d'une transaction. Une classe jobplace.sql.Transaction pourrait être créée.
- Implémenter la partie employé d'agence. Celui-ci peut éventuellement jouer un rôle de contrôle dans l'envoi des CV aux recruteurs ou des offres d'emploi aux candidats pour optimiser le processus. On peut aussi imaginer un système entièrement électronique où le rôle de l'employé d'agence est joué par un robot intelligent.
- L'envoi automatique par e-mails des offres d'emploi et des CV aux utilisateurs. L'utilisation de l'API JavaMail est possible.
- Mettre en place un système de sécurisation des informations. En effet, les codes d'accès ne constituent généralement qu'une protection illusoire.
    - Il serait possible d'implémenter des outils de détection d'intrusions qui alertent l'utilisateur et l'administrateur des données en cas d'anomalie.
    - Pour un stockage sécurisé, il serait possible de faire le cryptage des données. En ce qui concerne le profil utilisateur, des services d'annuaire comme LDAP pourraient être utilisés.
    - Pour la transmission des données, le *Secure Sockets Layer* (SSL) pourrait être utilisé. Pour utiliser SSL avec Tomcat 4, il faut activer le connecteur SSL.
- Utiliser la technique de filtres (spécification Java Servlets 2.3). Il serait possible de stocker les données dans un format XML. Un filtre permettrait à l'aide de XSLT de transformer les données pour avoir un format de sortie quelconque (HTML, WML, PDF).

9 Glossaire

Action	Spécification d'une instruction exécutable qui forme une abstraction d'une procédure de calcul. Une action résulte en un changement dans l'état du modèle et est réalisée par l'envoi d'un message à un objet ou en modifiant une valeur d'un attribut.
Application	Ensemble logiciel complet doté d'une finalité.
Architecture	La structure organisationnelle d'un système. Une architecture peut être décomposée de manière récursive en parties qui interagissent à travers les interfaces.
API	*Application Programming Interfaces*: bibliothèques de classes.
Attribut	Propriété nommée dans un classificateur qui décrit un ensemble valeurs que les instances du classificateur peuvent prendre.
Basedir	Répertoire principal.
C	Langage de programmation impératif, son utilisation est très répandue.
CGI	*Common Gateway Interface*: mécanisme utilisé par les serveurs Web pour communiquer avec les programmes externes. Cette communication génère des données dynamiques.
Classe	Description d'un ensemble d'objets qui partagent les mêmes attributs, opérations, méthodes, relations et sémantiques. Une classe peut utiliser un ensemble d'interfaces pour spécifier des collections d'opérations qu'elle fournit à son environnement.
Classificateur	Mécanisme qui décrit des caractéristiques de comportement et de structure. Les classificateurs incluent les interfaces, les classes, les types de données et les composants.
Classpath	Chemin utilisé par le compilateur Java pour trouver des classes nécessaires à la compilation.
Composant	Module ayant une identité et une interface bien définies.
Conteneur	Instance qui existe pour contenir d'autres instances et qui fournit des opérations pour accéder ou itérer son contenu.
Contexte	Vue d'un ensemble d'éléments de modèle en relation pour un objectif particulier, tel que spécifier une opération.
COMMIT	Méthode permettant d’effectuer une transaction.

Cookie	Petits bouts d'informations stockés sur le disque dur du client à la demande du serveur. Ils sont situés dans un fichier nommé `Cookie.txt`.
Cvthèque	Base de données contenant des CV.
E-mail	Courrier électronique sur Internet permettant d'échanger des messages entre utilisateurs éloignés.
E-recrutement	Recrutement en ligne.
Framework	Micro-architecture qui fournit une structure type pour les applications à l'intérieur d'un domaine précis.
Garbage collector	Ramasseur de déchets.
HTTP	*Hypertext Transfer Protocole*: protocole autorisant un document à circuler depuis le serveur Web jusqu'au navigateur.
Instance	Entité à laquelle peut s'appliquer un ensemble d'opérations et dont un état contient les résultats des opérations.
Interface	Collection d'opérations qui peut être utilisée pour décrire un service proposé par une instance.
Intranet	Réseau informatique interne d'une entreprise utilisant le même protocole que l'Internet.
Mapping	sorte de carte qui définit des liens entre différentes actions.
MIME	*Multipurpose Internet Mail Extension*: sert à typer les documents sur le Web.
Multithread	Un *thread* est un petit fil d'exécution autonome. *Multithread* signifie qui a plusieurs processus simultanément.
LDAP	*Lightweight Directory Access Protocol*: définit comment les clients doivent accéder aux données du serveur.
Lien complexe-complexe	Une liaison est dite complexe-complexe lorsque son degré (paire de types d'associations) est constitué de deux types d'associations complexes. Les cas possibles sont (m,m), (m,mc), (mc,m) et (mc,mc).
Lien simple-complexe	Une liaison est simple-complexe lorsque son degré consiste en un type d'association simple (type 1 ou c) et un type d'association complexe (type m ou mc).
Lien simple-simple	Une liaison est dite simple-simple lorsqu'elle est constituée de deux types d'associations simples ou conditionnelles. Les cas possibles sont (1,1), (1, c), (c, 1) et (c, c).

Open source	Le code source est mis à disposition et il est possible de le modifier.
Paquetage	Mécanisme général pour organiser des classes dans des groupes. Les paquetages peuvent s'emboîter à l'intérieur d'autres paquetages.
Paramètre	Spécification d'une variable qui peut être changée, passée ou envoyée.
Portabilité	Aptitude d'un système ou d'un logiciel à fonctionner sur différents systèmes d'exploitation et sur différents matériels.
Protocole	Ensemble de conventions définissant la transmission d'informations. Le protocole d'Internet s'appellent TCP/IP.
Query-by-example	Créer une requête à l'aide des exemples.
ROLLBACK	Méthode qui annule tous les changements depuis le dernier appel de COMMIT. Elle annule une transaction.
Servlets	***Serv****er-side app****let***: programme Java exécuté sur un serveur qui retourne une page HTML de résultat.
Smalltalk	langage de programmation.
SSL	*Secure Sockets Layer*: protocole de sécurité sur Internet le plus connu. Il établit une session sûre pour transmettre les données entre le client et le serveur.
Tag	Balise.
Thread	Voir *multithread*.
Transaction	permet de garantir qu'un ensemble d'opérations s'exécute selon le principe de l'atomicité.
URI	*Unified Resource Identifier*: identifiant d'un fichier.
URL	*Uniform Resource Locator*: est l'adresse d'un fichier sur Internet.
Web	*World Wide Web* ou WWW: réseau de documents hypermédias sur l'Internet qui sont reliés par des liens hypertextuels.
WAP	*Wireless Application Protocol*: protocole d'Internet sans fil.
WML	*Wireless Markup Language*: langage spécifié par WAP pour créer le contenu des applications WAP.

XML	*e**X**tensible **M**arkup **L**anguage*: c'est un langage permettant d'ajouter une signification à n'importe quelle portion de données. Ces éléments individuels peuvent être écrits, en les encadrant de balises appropriées.
XSLT	*e**X**tended **S**tylesheet **L**anguage **T**ransformation*: c'est un langage permettant de spécifier des règles de transformation sur un document XML dans le but de générer un document cible dérivé du document source XML.

10 Bibliographie

Livres

[Sharma/Sharma 2000] Sharma, Vivek; Sharma, Rajiv: Développement de sites e-commerce. CampusPress, Paris 2000.

[Ahmed et al. 2001] Ahmed, Kal, et al.: Professional Java XML. Wrox Press, Birmingham 2001.

[Avedal et al. 2000] Avedal, Karl et al: JSP Professionnel. Wrox Press et Editions Eyrolles, Paris 2001.

[Brown et al. 2001] Brown, Simon et al.: Professional JSP 2nd Edition. Wrox Press LtD, Birmingham 2001.

[Deitel/Deitel 2000] Deitel, H., M.; Deitel, P. J.: Comment programmer en java, troisième édition. Editions Reynald Gouled inc., Repentigny 2000.

[Dhoquois 2000] Dhoquois, Anne: Trouver un Emploi sur Internet. L'Etudiant 2001.

[Aboukhaled/Vanoirbeek 2000-2001] Aboukhaled, Omar; Vanoirbeek, Christine: Documents multimedia. Cours Université de Fribourg, 2000-2001.

[Alhir 1998] Alhir, Sinan Si: UML in a nutshell. O'Reilly, September 1998.

[Fannader/Leroux 1999] Fannader, Rémy; Leroux, Hervé: UML Principes de modélisation. Dunod, Paris 1999.

[Kettani et al. 1998] Kettani, Nasser et al.: De Merise à UML. Eyrolles, Paris 1998.

[Meier 2001] Meier, Andreas: Relationale Datenbanken. Springer-Verlag, 4. Aufl., Berlin, Heidelberg 2001.

[Atlan-Landaburu 2000] Atlan-Landaburu, Nathalie: 10 outils clés du cyber recruteur. Go Editions, Paris 2000.

Pages Web

[AGBS 2002] AGBS: Emploi sur Internet. Disponible: http://www.jobscout24.ch, accédé en janvier 2002.

[AltaVista Company 2001] AltaVista: Technologie de recherche. Disponible: http://www.altavista.com, accédé en décembre 2001.

[Apache 2002] Apache Software Foundation: Open source solution on the Java platform. Available: http://www.jakarta.apache.org, accessed January 2002.

[AVETH/ACIDE 2002] Ecoles Polytechniques Fédérales de Zurich et de Lausanne: Bourse électronique d'emplois. Disponible: http://www.telejob.ch, accédé en janvier 2002.

[Bytix 2002] Bytix AG : Banking and finance job exchange. Available: http://www.mcjob.ch, accédé en janvier 2002.

[Cadre Emploi 2002] Cadre Emploi: Recherche d'emploi. Disponible: http://www.cadreemploi.fr, accédé en janvier 2002.

[Carnegie Mellon University 2001] Carnegie Mellon University: Moteur de recherche. Disponible: http://www.lycos.com, accédé en décembre 2001.

[Copernic 2002] CopernicTechnologies inc.: Solutions de métarecherche. Disponible: http://www.copernic.com, accédé en janvier 2002.

[Edipresse/Publigroupe 2002] Edipresse et Publigroupe: Service emploi. Disponible: http://www.zannonces.ch, accédé en janvier 2002.

[EmailJob 2002] Emailjob: Recherche d'emploi. Disponible: http://www.emailjob.com, accédé en janvier 2002.

[Emploi 2002] Emploi: Journal interactif suisse. Disponible: http://www.emploi.ch, accédé en janvier 2002.

[Fast Search and Transfer ASA 2002] Fast Search and Transfer ASA: Moteur de recherche. Disponible: http://www.alltheweb.com, accédé en janvier 2002.

[Google 2001] Google: Gamme complète des services de recherche. Disponible : http://www.google.com, accédé en décembre 2001.

[Idealjob 2002] IdealJob Internet SA : Emploi sur Internet. Disponible : http://www.idealjob.ch, accédé en janvier 2002.

[INPS 2002] INPS: Info Space : Moteur des moteurs de recherche. Disponible: http://www.metacrawler.com, accédé en janvier 2002.

[jobpilot 2002] jobpilot : Offres d’emploi sur Internet en Europe. Disponible : http://www.jobpilot.ch, accédé en janvier 2002.

[La Liberté 2002] La Liberté: Quotidien romand édité à Fribourg. Disponible : http://www.laliberte.ch, accédé en janvier 2002.

[Limited 2001] Limited : Top jobs on the net. Available: http://www.topjobs.ch , accessed December 2001.

[Log4j] Apache Software Foundation: Projet open source pour le débogage et le logging. Disponible: http://jakarta.apache.org/log4j, accédé en janvier 2002.

[Monster 2002] Monster.com : Emploi sur Internet. Disponible: http://www.monster.ch, accédé en janvier 2002.

[MySQL 2002] MySQL AB: Open source database. Available: http://www.mysql.com, accessed January 2002.

[Nexus 2002] Nexus Personal- & Unternehmensberatung AG: Offres d'emploi. Disponible: http://www.nexus.ch, accédé en janvier 2002.

[persönlich 2002] persönlich Verlags AG: Portail suisse pour l'économie de communication. Disponible: http://www.persoenlich.com, accédé en janvier 2002.

[RIM 2001] RIM: Die Räber Information Management GmbH: The swiss search engine. Available: http://www.search.ch, accessed December 2001.

[Science Com 2002] Science Com SA: Le portail de la science et de l'innovation en Suisse. Disponible: http://www.swiss-science.org, accédé en janvier 2002.

[Search11 2002] Search11.com ltd: Moteur de recherche. Disponible: http://www.searche11.ch, accédé en janvier 2002.

[seco 2001] Le secrétariat d'Etat à l'économie: Information sur le marché du travail. Disponible: http://www.seco-admin.ch, accédé en décembre 2001.

[Skillworld 2002] Skillworld.com: Emploi en Suisse romande. Disponible: http://www.skillworld.com, accédé en janvier 2002.

[Sleepycat 2002] Sleepycat Software, Inc : Open Source embedded database technology. Available: http://www.sleepycat.com/, accessed February 2002.

[Stepstone 2002] Stepstone Deutschland AG : Recrutement en ligne. Disponible: http://www.stepstone.ch, accédé en janvier 2002.

[Struts 2002] Apache Software Foundation: Open source framework for building Web applications. Available: http://jakarta.apache.org/struts/index.html, accessed January 2002.

[Succes & Career 2002] Success & Career: Recherche d'emploi. Disponible: http://www.success-and-career.ch, accédé en janvier 2002.

[Sun Microsystems 2002] Sun Microsystems, Inc: Technologies Java. Disponible: http://java.sun.com, accédé en janvier 2002.

[SwebIndex 2001] SwebIndex : Swiss web index. Available: http://www.yoodle.ch, accessed December 2001.

[Swissclick 2002] Swissclick AG : Recherche d’emploi. Disponible: http://www.jobclick.ch, accédé en janvier 2002.

[Swift Management 2002] Swift Management AG: Recherche d'emploi pour étudiants et nouveaux diplômés. Disponible: http://www.diplom.ch, accédé en janvier 2002.

[Talents 2002] Talents: Career manager. Available: http://www.talents.ch, accessed January 2002.

[Themen park 2001] Themen park GmbH : Offres d’emploi. Disponible: http://www.worldwidejobs.com, accédé en décembre 2001.

[University of St. Gallen 2002] University of St. Gallen: Career Services Center. Available: http://www.vip.unigs.ch, accédé en janvier 2002.

[Weblaw 2002] Weblaw GmbH Cybersquare: Recherche d’emploi. Disponible: http://www.lawjobs.ch, accédé en janvier 2002.

[WinnerMarket 2002] WinnerMarket AG: Recherche d'emploi en informatique et banque. Disponible: http://www.jobwinner.ch, accédé en janvier 2002.

Revue

[Recto-Verso 2001] Recto-Verso Communication: Starting-Block, Nov./Déc.2001/n° 2, pp.60-69.

Annexe A: Tables

A.1: createtablesUsers.sql

```
create table users (
  user_name varchar(15) not null primary key,
  user_pass varchar(15) not null
);

create table user_roles (
  user_name varchar(15) not null,
  role_name varchar(15) not null,
  primary key (user_name, role_name)
);
```

A.2: createtablesJobplace.sql

```
CREATE TABLE users (
       user_id integer not null auto_increment,
       user_name VARCHAR(25),
       user_pass VARCHAR(25),
       user_role VARCHAR(25),
       primary key(user_id))
       type=BDB;

CREATE TABLE recruteur (
       societe_nom VARCHAR(100),
       societe_secteur_activite VARCHAR(100),
       societe_phone VARCHAR(25),
       societe_url VARCHAR(25),
       societe_description VARCHAR(250),
       contact_titre VARCHAR(25),
       contact_nom VARCHAR(25),
       contact_prenom VARCHAR(25),
       contact_email VARCHAR(25),
       fk_user_id integer not null,
       foreign key(fk_user_id) references users)
        type=BDB;

CREATE TABLE candidat (
       candidat_titre VARCHAR(25),
       candidat_prenom VARCHAR(25),
       candidat_nom VARCHAR(25),
       candidat_email VARCHAR(25),
       candidat_adresse VARCHAR(100),
       candidat_code_postal integer,
       candidat_ville VARCHAR(25),
       candidat_pays VARCHAR(25),
       candidat_date_naissance VARCHAR(25),
       candidat_phone VARCHAR(25),
       candidat_portable VARCHAR(25),
       fk_user_id integer not null,
       foreign key(fk_user_id) references users)
       type=BDB;

CREATE TABLE offre_emploi (
       offre_emploi_id integer not null auto_increment,
       offre_emploi_date date,
       offre_emploi_poste VARCHAR(50),
       offre_emploi_description_poste VARCHAR(255),
       offre_emploi_qualites_requises VARCHAR (100),
       offre_emploi_connaissances_techniques VARCHAR(100),
       offre_emploi_region VARCHAR(50),
       fk_recruteur_id integer not null,
       primary key(offre_emploi_id),
       foreign key(fk_recruteur_id)REFERENCES recruteur)
```

```
       type=BDB;

CREATE TABLE cv (
       cv_id integer not null auto_increment,
       cv_date date,
       cv_objectif VARCHAR(30),
       fk_candidat_id integer not null,
       primary key(cv_id),
       foreign key(fk_candidat_id) references candidat)
       type=BDB;

CREATE TABLE lettre_motivation (
       lettre_motivation_id integer not null auto_increment,
       lettre_motivation_texte varchar(255),
       fk_candidat_id integer not null,
       primary key(lettre_motivation_id),
       foreign key(fk_candidat_id) references candidat)
       type=BDB;

CREATE TABLE candidat_offre_emploi (
       fk_candidat_id integer not null,
       fk_offre_emploi_id integer not null,
       foreign key(fk_candidat_id) references candidat,
       foreign key(fk_offre_emploi_id) references offre_emploi,
       primary key(fk_offre_emploi_id, fk_candidat_id))
       type=BDB;

CREATE TABLE recruteur_cv (
       fk_recruteur_id integer not null,
       fk_cv_id integer not null,
       foreign key(fk_recruteur_id) references recruteur,
       foreign key(fk_cv_id) references cv,
       primary key(fk_recruteur_id, fk_cv_id))
       type=BDB;

CREATE TABLE formation (
       formation_id integer not null auto_increment,
       etablissement_nom varchar(25),
       formation_duree varchar(25),
       diplome_nom varchar(25),
       diplome_lieu varchar (25),
       fk_cv_id integer not null,
       primary key(formation_id),
       foreign key(fk_cv_id) references cv)
       type=BDB;

CREATE TABLE connaissances_informatiques (
       informatique_id integer not null auto_increment,
       logiciels varchar(25),
       langages varchar(25),
       systemes_exploitations varchar(25),
       programmation varchar(25),
       primary key(informatique_id),
       fk_cv_id integer not null,
       foreign key (fk_cv_id) references cv)
       type=BDB;

CREATE TABLE experience_professionnelle (
       experience_id integer not null auto_increment,
       entreprise_nom varchar(25),
       secteur_activite varchar(25),
       poste varchar(25),
       region varchar(25),
       debut_travail varchar(25),
       fin_travail varchar(25),
       competences varchar(25),
       fk_cv_id integer not null,
       primary key(experience_id),
       foreign key(fk_cv_id) references cv)
       type=BDB;
```

```
CREATE TABLE connaissances_linguistiques (
       langue_id integer not null auto_increment,
       langue_nom varchar(25),
       langue_niveau varchar(25),
       fk_cv_id integer not null,
       primary key(langue_id),
       foreign key(fk_cv_id) references cv)
       type=BDB;

CREATE TABLE postulation (
       postulation_id integer not null auto_increment,
       postulation_date date,
       fk_cv_id integer not null,
       fk_lettre_motivation_id integer not null,
       fk_offre_emploi_id integer not null,
       primary key(postulation_id),
       foreign key(fk_cv_id) references cv,
       foreign key(fk_lettre_motivation_id) references lettre_motivation,
       foreign key(fk_offre_emploi_id) references offre_emploi)
       type=BDB;
```

Annexe B: Fichiers JSP

B.1: accueil.jsp

```
<%@ page language="java" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<html:html locale="true">
<head>
<title><bean:message key="accueil.title"/></title>
<html:base/>
<link rel="stylesheet" href="/jobplace/pages/style.css" type="text/css">
</head>
<body leftmargin="0" topmargin="0" marginheight="0" marginwidth="0">
 <table height="65" width="100%" cellspacing="0" cellpadding="0" border="0" >
  <tr>
   <td valign="top" align="left" width="170"
background="/jobplace/pages/images/menupix.gif"><img
src="/jobplace/pages/images/menuhaut1.gif"border="0"/>
   </td>
   <td width=100%><img src="/jobplace/pages/images/menuhaut2.gif" border="0"/>
   </td>
  </tr>
 </table>
 <table width="100%" height="90%" cellspacing="0" cellpadding="0" border="0">
  <tr>
   <td width="170" align="left" valign="top"
background="/jobplace/pages/images/menupix.gif" height="100%"><br>
    <table width="170" cellspacing="0" cellpadding="0" border="0">
     <tr>
      <td background="/jobplace/pages/images/menu.gif" width="170" height="318"
align="left" valign="top" >
       <table class="fond" width="170" cellspacing="0" cellpadding="0" border="0">
       <tr>
        <td width="10" height="30"> </td>
        <td width="90"> </td>
       </tr>
       <tr>
        <td> </td>
        <td align="left"><font size="2"><b><font face="Arial, Helvetica, sans-
serif"><html:link page="/pages/accueil.jsp" styleClass="home"><bean:message
key="menu.accueil"/></html:link></font></b></font></td>
       </tr>
       <tr>
        <td> </td>
        <td align="left"><font size="2"><b><font face="Arial, Helvetica, sans-
serif"><html:link page="/pages/espace-candidat/espace-candidat.jsp"
styleClass="leftmenu"><bean:message
key="menu.espace_candidat"/></html:link></font></b></font></td>
       </tr>
       <tr>
        <td> </td>
        <td align="left"><font size="2"><b><font face="Arial, Helvetica, sans-
serif"><html:link page="/pages/espace-recruteur/espace-recruteur.jsp"
styleClass="leftmenu"><bean:message
key="menu.espace_recruteur"/></html:link></font></b></font></td>
       </tr>
       <tr>
        <td> </td>
        <td align="left"><font face="Arial, Helvetica, sans-serif"
size="2"><b><html:link page="/pages/conseil.jsp"
styleClass="leftmenu"><bean:message
key="menu.conseil"/></html:link></b></font></td>
       </tr>
       <tr>
```

```
       <td> </td>
       <td align="left"><font face="Arial, Helvetica, sans-serif"
size="2"><b><html:link page="/pages/contact.jsp"
styleClass="leftmenu"><bean:message
key="menu.contact"/></html:link></b></font></td>
      </tr>
      <tr>
       <td> </td>
       <td align="left" height="100%"> </td>
      </tr>
      </table>
     </td>
    </tr>
   </table>
  </td>
  <td valign="top" height="100%" width="100%">
   <table width="481" border="0" cellpadding="0" cellspacing="0"
mm:layoutgroup="true">
    <tr>
     <td width="24" height="9"></td>
     <td width="16"></td>
     <td width="171"></td>
     <td width="23"></td>
     <td width="171"></td>
     <td width="76"></td>
    </tr>
    <tr>
     <td height="41"></td>
     <td colspan="5" valign="top"><bean:message
key="accueil.heading"/><html:errors/> </td>
    </tr>
    <tr>
     <td height="30"></td>
     <td></td>
     <td></td>
     <td></td>
     <td></td>
     <td></td>
    </tr>
    <tr>
     <td height="117"></td>
     <td></td>
     <td valign="top"><html:link page="/pages/espace-candidat/espace-candidat.jsp"
styleClass="leftmenu">
      <html:img pageKey="espace-candidat.gif" border="0" width="171"
height="117"/>
      </html:link></td>
     <td></td>
     <td valign="top"><html:link page="/pages/espace-recruteur/espace-
recruteur.jsp" styleClass="leftmenu">
      <html:img pageKey="espace-recruteur.gif" border="0" width="171"
height="117"/>
      </html:link></td>
     <td></td>
    </tr>
    <tr>
     <td height="20"></td>
     <td></td>
     <td></td>
     <td></td>
     <td></td>
     <td></td>
    </tr>
    <tr>
     <td height="117"></td>
     <td></td>
     <td valign="top"><html:link page="/pages/conseil.jsp" styleClass="leftmenu">
      <html:img pageKey="conseil.gif" border="0" width="171" height="117"/>
      </html:link></td>
     <td></td>
```

```
      <td valign="top"><html:link page="/pages/contact.jsp" styleClass="leftmenu">
       <html:img pageKey="contact.gif" border="0" width="171" height="117"/>
       </html:link></td>
      <td></td>
     </tr>
    </table>
   </td>
  </tr>
  <tr>
   <td align="left" height="15"><html:img page="/pages/images/menuarrondi.jpg"
width="170" height="15" border="0"/></td>
   <td valign="top" background=""></td>
  </tr>
 </table>
 <table width="100%" cellspacing="0" cellpadding="0" border="0" valign="top">
  <tr>
   <td width="749" height="21"><html:img pageKey="menubas" width="749" height="21"
usemap="#menu_basMap" border="0"/></td>
   <td background="" width="100%" rowspan="2"> </td>
  </tr>
  <tr>
   <td width="749" height="21" align="left"
background="/jobplace/pages/images/menubas2.gif"><font face="arial" size="1"
color="#0066CC">  <bean:message
key="realisation"/>          &nbs
p;             &nb
sp;             &n
bsp;             &
nbsp;            <bean:
message key="jobplace.adresse"/></font></td>
  </tr>
 </table>
 <map name="menu_basMap">
  <area shape="rect" coords="532,5,581,18" href="/jobplace/pages/contact.jsp"
target="_parent">
  <area shape="rect" coords="465,5,514,18" href="/jobplace/pages/conseil.jsp"
target="_parent">
  <area shape="rect" coords="345,5,450,18" href="/jobplace/pages/espace-
recruteur/espace-recruteur.jsp" target="_parent">
  <area shape="rect" coords="228,5,325,18" href="/jobplace/pages/espace-
candidat/espace-candidat.jsp" target="_parent">
  <area shape="rect" coords="166,5,211,18" href="/jobplace/pages/accueil.jsp"
target="_parent">
 </map>
</body>
</html:html>
```

B.2: extrait de logon.jsp

```
<bean:message key="logon.heading"/>
<form method="POST" action='<%= response.encodeURL("j_security_check") %>'>
  <table border="0" cellspacing="5">
    <tr>
      <td align="right"><bean:message key="prompt.user_name"/></td>
      <td align="left"><input type="text" name="j_username"></td>
    </tr>
    <tr>
      <td align="right"><bean:message key="prompt.user_pass"/></td>
      <td align="left"><input type="password" name="j_password"></td>
    </tr>
    <tr>
      <td align="right">
        <html:submit>
          <bean:message key="button.logon.submit"/>
        </html:submit>
      </td>
      <td align="left">
        <html:reset>
          <bean:message key="button.reset"/>
```

```
        </html:reset>
        <html:cancel>
          <bean:message key="button.cancel"/>
        </html:cancel>
      </td>
    </tr>
  </table>
</form>
```

B.3: extrait de caddy.jsp

```
<table cellspacing="2" cellpadding="2" border="1">
 <logic:equal name="<%= Constants.CADDY_KEY %>" property="kind"
                    scope="session" value="vide">
  <bean:message key="caddy.empty"/>
 </logic:equal>
 <logic:equal name="<%= Constants.CADDY_KEY %>" property="kind"
                    scope="session" value="offre">
  <tr>
   <td><bean:message key="offre_emploi_date" /></td>
   <td><bean:message key="offre_emploi_poste" /></td>
   <td><bean:message key="offre_emploi_region" /></td>
   <td><bean:message key="offre_emploi_ref" /></td>
   <td> </td>
   <td> </td>
  </tr>
  <logic:iterate id="caddyItem" type="jobplace.common.CaddyItem"
              name="<%= Constants.CADDY_KEY %>" property="caddyItems">
   <tr>
    <td><bean:write name="caddyItem" property="offre_emploi_date"/></td>
    <td><bean:write name="caddyItem" property="offre_emploi_poste"/></td>
    <td><bean:write name="caddyItem" property="offre_emploi_region"/></td>
    <td><bean:write name="caddyItem" property="offre_emploi_id"/></td>
    <td><html:link page="/do/supprimer" styleClass="normal" name="caddyItem"
property="mapping">
       <bean:message key="caddy.delete"/></html:link></td>
    <td><html:link page="/do/preparePostulation" styleClass="normal"
name="caddyItem" property="mapping">
       <bean:message key="postulation.action"/></html:link></td>
   </tr>
  </logic:iterate>
 </logic:equal>
 <logic:equal name="<%= Constants.CADDY_KEY %>" property="kind"
                    scope="session" value="cv">
  <tr>
   <td><bean:message key="offre_emploi_date" /></td>
   <td><bean:message key="offre_emploi_poste" /></td>
   <td><bean:message key="offre_emploi_region" /></td>
   <td><bean:message key="offre_emploi_ref" /></td>
  </tr>
  <logic:iterate id="caddyItem" type="jobplace.common.CaddyItem"
      name="<%= Constants.CADDY_KEY %>" property="caddyItems">
  </logic:iterate>
 </logic:equal>
</table>
```

B.4: extrait de error.jsp

```
<td valign="top" height="100%" width="100%">
  <bean:message key="error_page.heading"/>
  <blockquote><html:errors/></blockquote>
  <form>
    <div align="center"><center><p><input type="button" value=" Back "
onClick="history.go(-1)"></p></center></div>
    </form>
</td>
```

B.5: extrait de espace-candidat.jsp

```
<td valign="top" height="100%" width="100%">
  <html:errors/>
  <bean:message key="espace-candidat.heading"/>
  <ul>
    <li>
      <html:link page="/do/editInscription?action=Create&user_role=candidat"
styleClass="normal">
        <bean:message key="espace-candidat.registration"/>
      </html:link>
    </li>
    <li>
      <html:link page="/do/ShowOffres" styleClass="normal">
        <bean:message key="espace-candidat.consultation-offres"/>
      </html:link>
    </li>
    <li>
      <html:link page="/pages/espace-candidat/compte-candidat/compte-candidat.jsp"
styleClass="normal">
        <bean:message key="espace-candidat.compte-candidat"/>
      </html:link>
    </li>
  </ul>
</td>
```

B.6: extrait de showOffre.jsp

```
<td valign="top" height="100%" width="100%">
  <html:errors/>
  <bean:message key="showOffre.heading"/>
  <table border="1">
    <tr>
      <td>
        <bean:message key="offre_emploi_ref"/>
      </td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_id"/></td>
    </tr>
    <tr>
      <td><bean:message key="offre_emploi_date"/></td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_date"/></td>
    </tr>
    <tr>
      <td><bean:message key="offre_emploi_poste"/></td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_poste"/></td>
    </tr>
    <tr>
      <td><bean:message key="offre_emploi_description_poste"/></td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_description_poste"/></td>
    </tr>
    <tr>
      <td><bean:message key="offre_emploi_qualites_requises"/></td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_qualites_requises"/></td>
    </tr>
    <tr>
      <td><bean:message key="offre_emploi_connaissances_techniques"/>
      </td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_connaissances_techniques"/></td>
    </tr>
    <tr>
      <td><bean:message key="offre_emploi_region"/></td>
      <td><bean:write name="<%= Constants.OFFRE_EMPLOI_KEY %>"
property="offre_emploi_region"/></td>
```

```
    </tr>
  </table>
  <p></p>
  <html:link page="/do/addToCaddy?input_path=showOffre" name="<%=
Constants.OFFRE_EMPLOI_KEY %>" property="mapping">
    <bean:message key="caddy.add"/>
  </html:link><br>
  <html:link page="/do/preparePostulation" styleClass="normal" name="<%=
Constants.OFFRE_EMPLOI_KEY %>" property="mapping">
    <bean:message key="postulation.action"/>
  </html:link><br>
  <html:link page="/pages/caddy.jsp"><bean:message key="caddy.show"/>
  </html:link>
</td>
```

B.7: extrait de espace-candidat/consultation-offres.jsp

```
<bean:message key="consultation-offres.heading"/>
<bean:message key="consultation-offres.text"/>
  <p></p>
  <table border="1">
    <tr>
       <td width="60"><bean:message key="offre_emploi_date"/></td>
       <td width="100"><bean:message key="offre_emploi_poste"/></td>
       <td width="100"><bean:message key="offre_emploi_region"/></td>
       <td width="20"><bean:message key="offre_emploi_ref"/></td>
       <td> </td>
    </tr>
    <logic:iterate id="offreemploi" type="jobplace.offre.OffreEmploi"
       name="<%= Constants.OFFRES_EMPLOI_ARRAY_KEY %>">
       <tr>
         <td><bean:write name="offreemploi" property="offre_emploi_date"/>
        </td>
        <td><html:link page="/do/GetOffre" name="offreemploi" property="mapping">
            <bean:write name="offreemploi" property="offre_emploi_poste"/>
          </html:link>
        </td>
         <td><bean:write name="offreemploi" property="offre_emploi_region"/>
         </td>
         <td><bean:write name="offreemploi" property="offre_emploi_id"
filter="true"/></td>
         <td><html:link page="/do/addToCaddy?input_path=consultation-offres"
name="offreemploi" property="mapping">
             <bean:message key="caddy.add"/></html:link></td>
      </tr>
    </logic:iterate>
  </table>
  <p></p>
  <html:link page="/pages/caddy.jsp"><bean:message key="caddy.show"/>
  </html:link>
```

B.8: extrait de selectLettreMotivation.jsp

```
<bean:message key="lettre-motivation.heading"/>
<bean:message key="lettre-motivation.text"/>
<p></p>
<table border="1">
  <tr>
    <td width="15"><bean:message key="lettre_motivation_id"/></td>
    <td width="25"><bean:message key="lettre_motivation_texte"/></td>
    <td> </td>
  </tr>
  <logic:iterate id="lettre" type="jobplace.lettremotivation.LettreMotivationForm"
name="<%= Constants.LETTRE_MOTIVATION_ARRAY_KEY %>">
    <tr>
      <td>
        <html:link page="/do/editLettreMotivation?action=Edit" name="lettre"
property="mapping"><bean:write name="lettre" property="lettre_motivation_id"/>
        </html:link>
```

```
      </td>
      <td><bean:write name="lettre" property="lettre_motivation_texte_tmp"/></td>
      <td><html:link page="/do/supprimerLettre" name="lettre" property="mapping">
        <bean:message key="lettre_motivation.supprimer"/></html:link>
      </td>
    </tr>
  </logic:iterate>
</table>
<p></p>
<html:link page="/do/editLettreMotivation?action=Create">
  <bean:message key="lettre_motivation.createNew"/>
</html:link>
```

B.9: extrait de CV.jsp

```
<!-- save is needed for both Create and Edit -->
<html:form action="/saveCV">
  <table align="center" cellspacing="0" cellpadding="0" border="0" width="480">
  <html:hidden property="action"/>
  <html:hidden property="candidat.candidat_id"/>
  <html:hidden property="cv_id"/>
  <html:hidden property="formation.formation_id"/>
  <html:hidden property="langues.langue_id"/>
  <html:hidden property="informatiques.informatique_id"/>
  <html:hidden property="profession.experience_id"/>
<!-- START Header: Create a new CV -->
<logic:equal name="CVForm" property="action"
             scope="request" value="Create">
  <tr>
    <td><bean:message key="cv.header1.create"/></td>
  </tr>
  <tr>
    <td><p><bean:message key="cv.create"/></p><p></p></td>
  </tr>
  <tr><td><bean:message key="cv.note"/></td></tr>
</logic:equal>
<!-- END Header: Create a new CV -->
<!-- START Header: Edit an existing CV -->
<logic:equal name="CVForm" property="action"
             scope="request" value="Edit">
  <tr>
    <td><bean:message key="cv.header1.edit"/></td>
  </tr>
  <tr>
    <td><p><bean:message key="cv.edit"/></p><p></p></td>
  </tr>
  <tr><td><bean:message key="cv.note"/></td></tr>
</logic:equal>
<!-- END Header: Edit an existing CV -->
<!-- Common to both Create and Edit -->
<tr>
  <table align="center" cellspacing="3" cellpadding="0" border="0" width="480">
   <tr>
    <td colspan="2"><h5><bean:message key="cv_objectif"/></h5></td>
    </tr>
   <tr>
     <td colspan="2">
       <html:text property="cv_objectif" size="34" maxlength="50"/>
     </td>
   </tr>
   <tr>
    <td colspan="2" height="20"></td>
   </tr>
   <tr>
    <td colspan="2"><h5><bean:message key="cv.profil"/></h5></td>
   </tr>
   <tr>
    <td width="120"><bean:message key="candidat.candidat_titre"/></td>
    <td>
```

```
     <html:select property="candidat.candidat_titre">
      <html:option key="titre.mister" value="Monsieur"/>
       <html:option key="titre.misses" value="Madame"/>
     </html:select>
    </td>
   </tr>
   <tr>
    <td width="120"><bean:message key="candidat.candidat_prenom"/></td>
    <td><html:text property="candidat.candidat_prenom" size="34"
maxlength="50"/></td>
   </tr>
   <tr>
    <td width="120"><bean:message key="candidat.candidat_nom"/></td>
    <td><html:text property="candidat.candidat_nom" size="34" maxlength="50"/></td>
   </tr>
```

B.10: extrait de LettreMotivation.jsp

```
<html:form action="/saveLettreMotivation">
  <table align="center" cellspacing="0" cellpadding="0" border="0" width="480">
    <html:hidden property="action"/>
    <html:hidden property="fk_candidat_id"/>
    <html:hidden property="lettre_motivation_id"/>
    <!-- START: Create a new lettre motivation -->
    <logic:equal name="LettreMotivationForm" property="action"
                    scope="request" value="Create">
      <tr>
        <td><bean:message key="lettre-motivation.heading.create"/></td>
       </tr>
       <tr>
         <td>
           <p><bean:message key="lettre-motivation.create"/></p><p></p>
         </td>
       </tr>
    </logic:equal>
    <!-- START: Edit lettre motivation -->
    <logic:equal name="LettreMotivationForm" property="action"
                    scope="request" value="Edit">
      <tr>
        <td><bean:message key="lettre-motivation.heading.edit"/></td>
       </tr>
       <tr>
         <td><p><bean:message key="lettre-motivation.edit"/></p><p></p></td>
       </tr>
    </logic:equal>
      <tr>
        <td>
          <html:textarea property="lettre_motivation_texte" cols="60"
rows="20"/></td>
       </tr>
       <tr>
         <td>
          <html:submit><bean:message key="button.submit"/></html:submit>
          <html:cancel><bean:message key="button.cancel"/></html:cancel>
         </td>
       </tr>
  </table>
</html:form>
```

B.11: extrait de postulation.jsp

```
<html:form action="/postuler">
  <html:hidden property="offre_emploi.offre_emploi_id"/>
  <html:hidden property="cvform.cv_id"/>
  <tr>
    <td><bean:message key="offre_emploi_poste"/></td>
    <td><bean:write name="PostulationForm"
property="offre_emploi.offre_emploi_poste"/>
    </td>
```

```
  </tr>
  <tr>
    <td><bean:message key="cv_objectif"/></td>
    <td><bean:write name="PostulationForm" property="cvform.cv_objectif"/>
    </td>
   </tr>
   <tr>
    <td colspan="2">
       <html:select property="lettreform.lettre_motivation_id">
         <logic:iterate id="lettremotivation"
type="jobplace.lettremotivation.LettreMotivationForm" name="<%=
Constants.LETTRE_MOTIVATION_ARRAY_KEY %>">
           <html:option value="<%=
Integer.toString(lettremotivation.getLettre_motivation_id()) %>" >
             <bean:write name="lettremotivation"
property="lettre_motivation_texte"/>
           </html:option>
         </logic:iterate>
       </html:select>
    </td>
  </tr>
  <tr>
    <td><html:submit><bean:message key="button.submit"/></html:submit></td>
    <td><html:cancel><bean:message key="button.cancel"/></html:cancel></td>
  </tr>
</html:form>
```

B.12: extrait de postulationOK.jsp

```
<td valign="top" height="100%" width="100%">
<html:errors/>
    <bean:message key="postulation.heading"/>
    <bean:message key="postulation.text_ok"/>
</td>
```

B.13: extrait de compte-candidat.jsp

```
<bean:message key="compte-candidat.heading"/>
<html:link page="/do/Logoff"><bean:message key="logoff"/></html:link><br>
<html:link page="/do/editCV?action=Create">
  <bean:message key="compte-candidat.cv.create"/>
</html:link> / 
<html:link page="/do/editCV?action=Edit">
  <bean:message key="compte-candidat.cv.edit"/>
</html:link> <bean:message key="compte-candidat.cv.texte"/><br>
<html:link page="/do/editInscription?action=Edit&user_role=candidat">
  <bean:message key="compte-candidat.edit-inscription"/>
</html:link><br>
<html:link page="/do/showLettres">
  <bean:message key="compte-candidat.lettremotivation.edit"/>
</html:link>
```

B.14: extrait de showCV.jsp

```
<bean:message key="showCV.heading"/>
<table border="0">
  <tr>
    <td><bean:message key="cv_objectif"/></td>
    <td><bean:write name="<%= Constants.CVFORM_KEY %>"
property="cv_objectif"/></td>
  </tr>
</table>
<table>
  <tr>
    <td>
       <html:link page="/do/addToCaddy?input_path=showCV" name="<%=
Constants.CVFORM_KEY %>" property="mapping">
         <bean:message key="caddy.add"/>
       </html:link>
```

```
    </td>
    <td>
       <html:link page="/pages/caddy.jsp"><bean:message key="caddy.show"/>
      </html:link>
    </td>
  </tr>
</table>
```

B.15: extrait de espace-recruteur.jsp

```
<bean:message key="espace-recruteur.heading"/>
  <ul>
    <li>
      <html:link page="/do/editInscription?action=Create&user_role=recruteur">
         <bean:message key="espace-recruteur.inscription"/></html:link>
    </li>
    <li>
      <html:link page="/pages/espace-recruteur/compte-recruteur/compte-
recruteur.jsp">
        <bean:message key="espace-recruteur.compte-recruteur"/></html:link>
    </li>
    <li>
       <html:link page="/do/ShowCVs">
         <bean:message key="espace-recruteur.consultation-cv"/></html:link>
    </li>
  </ul>
```

B.16: extrait de consultation-cvs.jsp

```
<bean:message key="consultation-cvs.heading"/>
<table border="1">
  <tr>
    <td><bean:message key="cv_objectif"/></td>
    <td><bean:message key="cv_id"/></td>
  </tr>
  <logic:iterate id="cv" type="jobplace.cv.CVForm" name="<%=
Constants.CVS_ARRAY_KEY %>">
    <tr>
      <td><html:link page="/do/GetCV" name="cv" property="mapping">
         <bean:write name="cv" property="cv_objectif"/></html:link>
      </td>
      <td><bean:write name="cv" property="cv_id"/></td>
      <td><html:link page="/do/addToCaddy?input_path=consultation-cvs" name="cv"
property="mapping"> <bean:message key="caddy.add"/></html:link>
      </td>
    </tr>
  </logic:iterate>
</table>
<html:link page="/pages/caddy.jsp"><bean:message key="caddy.show"/>
</html:link>
```

B.17: extrait de compte-recruteur/consultation-offres.jsp

```
<bean:message key="consultation-offres.heading"/>
<table border="1">
  <tr>
    <td><bean:message key="offre_emploi_date"/></td>
    <td><bean:message key="offre_emploi_poste"/></td>
    <td><bean:message key="offre_emploi_region"/></td>
    <td><bean:message key="offre_emploi_ref"/></td>
    <td> </td>
    <td> </td>
  </tr>
  <logic:iterate id="offreemploi" type="jobplace.offre.OffreEmploi"
                   name="<%= Constants.OFFRES_EMPLOI_ARRAY_KEY %>">
    <tr>
       <td><bean:write name="offreemploi" property="offre_emploi_date"/>
      </td>
       <td>
```

```
        <html:link page="/do/editOffre?action=Edit" name="offreemploi"
property="mapping">
          <bean:write name="offreemploi" property="offre_emploi_poste"/>
        </html:link>
       </td>
       <td><bean:write name="offreemploi" property="offre_emploi_region"/>
      </td>
       <td><bean:write name="offreemploi" property="offre_emploi_id"
filter="true"/></td>
       <td><html:link page="/do/supprimerOffre" name="offreemploi"
property="mapping"><bean:message key="offre_emploi.supprimer"/>
        </html:link></td>
      <td>
         <html:link page="/do/ShowPostulations" name="offreemploi"
property="mapping"><bean:message key="consultation-postulations.show"/>
        </html:link></td>
    </tr>
  </logic:iterate>
</table>
<p></p>
<html:link page="/do/editOffre?action=Create">
  <bean:message key="offre_emploi.new"/>
</html:link>
```

B.18: extrait de compte-recruteur.jsp

```
<bean:message key="compte-recruteur.heading"/>
<html:link page="/do/Logoff"><bean:message key="logoff"/></html:link><br>
<html:link page="/do/ShowOffresEdit">
  <bean:message key="compte-recruteur.offres"/></html:link><br>
<html:link page="/do/editInscription?action=Edit&user_role=recruteur">
  <bean:message key="compte-recruteur.edit-inscription"/>
</html:link><br>
<html:link page="/do/ShowOffresEdit">
   <bean:message key="compte-recruteur.postulations"/>
</html:link>
```

B.19: extrait de consultation-postulations.jsp

```
<bean:message key="consultation-postulations.heading"/>
<table border="1">
  <tr>
    <td><bean:message key="offre_emploi_date"/></td>
    <td><bean:message key="offre_emploi_poste"/></td>
    <td><bean:message key="offre_emploi_region"/></td>
    <td><bean:message key="offre_emploi_ref"/></td>
  </tr>
  <logic:iterate id="postulation" type="jobplace.postulation.PostulationForm"
name="<%= Constants.POSTULATIONS_ARRAY_KEY %>">
    <tr>
       <td><bean:write name="postulation" property="postulation_date"/></td>
       <td><html:link page="/do/GetPostulation" name="postulation"
property="mapping">
          <bean:write name="postulation" property="cvform.cv_objectif"/>
        </html:link>
       </td>
    </tr>
  </logic:iterate>
</table>
```

B.20: extrait de Offre.jsp

```
<!-- save is needed for both Create and Edit -->
<html:form action="/saveOffre">
  <table align="center" cellspacing="0" cellpadding="0" border="0" width="480">
    <html:hidden property="action"/>
    <html:hidden property="offre_emploi_id"/>
    <html:hidden property="fk_recruteur_id"/>
    <!-- START Header: Create a new Offre -->
```

```
    <logic:equal name="OffreForm" property="action" scope="request" value="Create">
      <tr>
        <td><bean:message key="offre.header1.create"/></td>
       </tr>
       <tr>
        <td><p><bean:message key="offre.create"/></p><p></p></td>
      </tr>
       <tr><td><bean:message key="offre.note"/></td></tr>
    </logic:equal>
    <!-- END Header: Create a new Offre -->
    <!-- START Header: Edit an existing Offre -->
    <logic:equal name="OffreForm" property="action" scope="request" value="Edit">
      <tr>
        <td><bean:message key="offre.header1.edit"/></td>
       </tr>
       <tr>
        <td><p><bean:message key="offre.edit"/></p><p></p></td>
      </tr>
       <tr><td><bean:message key="offre.note"/></td></tr>
    </logic:equal>
    <!-- END Header: Edit an existing Offre -->
    <!-- Common to both Create and Edit -->
    <tr>
       <table align="center" cellspacing="3" cellpadding="0" border="0" width="480">
         <tr>
           <td width="120"><bean:message key="offre_emploi_poste"/></td>
           <td><html:text property="offre_emploi_poste" size="30"
maxlength="50"/></td>
         </tr>
         <tr>
          <td width="120"><bean:message key="offre_emploi_description_poste"/></td>
           <td><html:textarea property="offre_emploi_description_poste"/>
          </td>
        </tr>
```

B.21: extrait de showPostulation.jsp

```
<bean:message key="showPostulation.heading"/>
<table>
  <tr>
    <td>
      <bean:message key="postulation.lettre_motivation.title"/>
      <font size=-1>
        <bean:write name="<%= Constants.POSTULATION_KEY %>"
property="lettreform.lettre_motivation_texte"/>
      </font>
      <p></p>
    </td>
  </tr>
  <tr>
    <td>
      <bean:message key="postulation.lettre_motivation.cv"/>
        <table>
          <tr>
            <td>
              <font size=-1>
                <bean:write name="<%= Constants.POSTULATION_KEY %>"
property="cvform.candidat.candidat_nom"/>
                <bean:write name="<%= Constants.POSTULATION_KEY %>"
property="cvform.candidat.candidat_prenom"/>
```

B.22: extrait de inscription.jsp

```
<!-- save is needed for both Create and Edit -->
<html:form action="/saveInscription">
<table border="0" width="480">
  <html:hidden property="action"/>
```

```
  <html:hidden property="user_role"/>
  <html:hidden property="candidat.candidat_id"/>
  <!-- Head: Create a new User -->
  <logic:equal name="InscriptionForm" property="action"scope="request"
value="Create">
    <tr>
      <td><bean:message key="inscription.header1.create"/></td>
    </tr>
    <tr>
      <td><p><bean:message key="inscription.create"/></p><p></p></td>
    </tr>
    <tr><td><bean:message key="inscription.note"/></td></tr>
    <tr>
      <table align="center" cellspacing="3" cellpadding="0" border="0" width="480">
        <tr>
          <td width="120"><bean:message key="inscription.user_name"/></td>
          <td><html:text property="user_name" size="25" maxlength="25"/></td>
        </tr>
        <tr>
          <td width="120"><bean:message key="inscription.user_pass"/></td>
          <td><html:password property="user_pass" size="25" maxlength="25"/></td>
        </tr>
  </logic:equal>
  <!-- Head: Edit User -->
  <logic:equal name="InscriptionForm" property="action" scope="request"
value="Edit">
    <tr>
      <td><bean:message key="inscription.header1.edit"/></td>
    </tr>
    <tr>
      <td><p><bean:message key="inscription.edit"/></p><p></p></td>
    </tr>
    <tr><td><bean:message key="inscription.note"/></td></tr>
    <tr>
      <table align="center" cellspacing="3" cellpadding="0" border="0" width="480">
  </logic:equal>
  <!-- START: Candidat -->
  <logic:equal name="InscriptionForm" property="user_role"scope="request"
value="candidat">
    <tr>
      <td width="120"><bean:message key="candidat.candidat_titre"/></td>
      <td><html:select property="candidat.candidat_titre">
          <html:option key="titre.mister" value="Monsieur"/>
          <html:option key="titre.misses" value="Madame"/>
        </html:select></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_prenom"/></td>
        <td><html:text property="candidat.candidat_prenom" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_nom"/></td>
        <td><html:text property="candidat.candidat_nom" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_email"/></td>
        <td><html:text property="candidat.candidat_email" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_adresse"/>
        </td>
        <td><html:text property="candidat.candidat_adresse" size="25"
maxlength="100"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_code_postal"/>
        </td>
```

```
        <td><html:text property="candidat.candidat_code_postal" size="10"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_ville"/></td>
        <td><html:text property="candidat.candidat_ville" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_pays"/></td>
        <td><html:text property="candidat.candidat_pays" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_date_naissance"/></td>
       <td><html:text property="candidat.candidat_date_naissance" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_phone"/></td>
        <td><html:text property="candidat.candidat_phone" size="25"
maxlength="50"/></td>
      </tr>
      <tr>
        <td width="120"><bean:message key="candidat.candidat_portable"/>
       </td>
        <td><html:text property="candidat.candidat_portable" size="25"
maxlength="50"/></td>
      </tr>
    </logic:equal>
    <!-- END: Candidat -->
```

Annexe C: Fichiers de configuration

C.1: style.css

```
body {background-color:white;font-family:Arial,Helvetica}
A.normal { color:black}
A.normal:active { color:#0066CC; font-style:italic}
A.normal:visited { color:#0066CC}
A.accueil { text-decoration:none;color:black}
A.accueil:hover { text-decoration:none;color:#0066CC}
A.accueil:visited.hover {text-decoration:none; color:#0066CC}
A.home { text-decoration:none; color:0066cc;}
A.home:hover { text-decoration:none; color:0066cc; }
A.home:visited.hover { text-decoration:none; color:0066cc;}
A.leftmenu { text-decoration:none; color:black; }
A.leftmenu:hover { text-decoration:none; color:0066cc; }
A.leftmenu:visited.hover { text-decoration:none; color:0066cc; }
table.fond td {background-image : inherit;}
```

C.2: struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">

<struts-config>

  <!-- ========== Form Bean Definitions ===================================-->
  <form-beans>

    <form-bean name="InscriptionForm"
               type="jobplace.inscription.InscriptionForm"/>
    <form-bean name="LogonForm"
               type="jobplace.logon.LogonForm"/>
    <form-bean name="CVForm"
               type="jobplace.cv.CVForm"/>
    <form-bean name="LettreMotivationForm"
               type="jobplace.lettremotivation.LettreMotivationForm"/>
    <form-bean name="PostulationForm"
              type="jobplace.postulation.PostulationForm"/>
    <form-bean name="OffreForm"
              type="jobplace.offre.OffreEmploi"/>

  </form-beans>


  <!-- ========== Global Forward Definitions ==============================-->
  <global-forwards>

     <forward
        name="espace-candidat"
        path="/pages/espace-candidat/compte-candidat/compte-candidat.jsp"/>
     <forward
        name="compte-candidat"
        path="/pages/espace-candidat/compte-candidat/compte-candidat.jsp"/>
     <forward
        name="espace-recruteur"
        path="/pages/espace-recruteur/compte-recruteur/compte-recruteur.jsp"/>
     <forward
        name="error"
        path="/pages/error.jsp"/>
     <forward
        name="cancel"
        path="/pages/accueil.jsp"/>
```

```
     <forward
        name="createCV"
        path="/do/editCV?action=Create"/>
     <forward
        name="createLettre"
        path="/do/editLettreMotivation?action=Create"/>
     <forward
        name="preparePostulation"
        path="/do/preparePostulation"/>

  </global-forwards>


  <!-- ========== Action Mapping Definitions ==============================-->
  <action-mappings>

    <action path="/editInscription"
            type="jobplace.inscription.editInscriptionAction"
            name="InscriptionForm"
            scope="request"
            validate="false">
            <forward name="continue" path="/pages/inscription/inscription.jsp"/>
    </action>
    <action path="/saveInscription"
            type="jobplace.inscription.saveInscriptionAction"
            name="InscriptionForm"
            scope="request"
            validate="false"
            input="/pages/inscription/inscription.jsp">
            <forward name="continue-candidat" path="/pages/espace-candidat/espace-
candidat.jsp"/>
            <forward name="continue-recruteur" path="/pages/espace-
recruteur/espace-recruteur.jsp"/>
    </action>
    <action path="/Logon"
            type="jobplace.logon.LogonAction"
            name="LogonForm"
            scope="request"
            validate="true"
            input="/pages/logon.jsp">
            <forward name="continue" path="/pages/espace-candidat/compte-
candidat/compte-candidat.jsp"/>
    </action>
    <action path="/saveCV"
            type="jobplace.cv.saveCVAction"
            name="CVForm"
            scope="request"
            validate="false"
            input="/pages/espace-candidat/compte-candidat/CV.jsp">
            <forward name="continue" path="/pages/espace-candidat/compte-
candidat/compte-candidat.jsp"/>
    </action>
    <action path="/editCV"
            type="jobplace.cv.editCVAction"
            name="CVForm"
            scope="request"
            validate="false"
            input="/pages/espace-candidat/compte-candidat/compte-candidat.jsp">
            <forward name="continue" path="/pages/espace-candidat/compte-
candidat/CV.jsp"/>
    </action>
    <action path="/showLettres"
            type="jobplace.lettremotivation.showLettresMotivationAction">
          <forward name="continue" path="/pages/espace-candidat/compte-
candidat/selectLettreMotivation.jsp"/>
    </action>
    <action path="/editLettreMotivation"
            type="jobplace.lettremotivation.editLettreMotivationAction"
            name="LettreMotivationForm"
            scope="request"
```

```
            validate="false"
            input="/pages/espace-candidat/compte-candidat/compte-candidat.jsp">
            <forward name="continue" path="/pages/espace-candidat/compte-
candidat/LettreMotivation.jsp"/>
    </action>
    <action path="/saveLettreMotivation"
            type="jobplace.lettremotivation.saveLettreMotivationAction"
            name="LettreMotivationForm"
            scope="request"
            validate="false"
            input="/pages/espace-candidat/compte-candidat/LettreMotivation.jsp">
            <forward name="continue" path="/do/showLettres"/>
    </action>
    <action path="/supprimerLettre"
            type="jobplace.lettremotivation.SupprimerAction">
          <forward name="continue" path="/do/showLettres"/>
    </action>
    <action path="/preparePostulation"
            type="jobplace.postulation.preparePostulationAction"
            name="PostulationForm"
            scope="request"
            validate="false">
            <forward name="continue" path="/pages/espace-candidat/compte-
candidat/postulation.jsp"/>
    </action>
    <action path="/postuler"
            type="jobplace.postulation.PostulationAction"
            name="PostulationForm"
            scope="request"
            validate="true">
            <forward name="continue" path="/pages/espace-candidat/compte-
candidat/postulationOK.jsp"/>
            <forward name="cancel" path="/pages/espace-candidat/compte-
candidat/compte-candidat.jsp"/>
    </action>
    <action path="/ShowPostulations"
            type="jobplace.postulation.ShowPostulationsAction">
          <forward name="continue" path="/pages/espace-recruteur/compte-
recruteur/consultation-postulations.jsp"/>
    </action>
    <action path="/GetPostulation"
            type="jobplace.postulation.getPostulationAction">
          <forward name="continue" path="/pages/espace-recruteur/compte-
recruteur/showPostulation.jsp"/>
    </action>
    <action path="/saveOffre"
            type="jobplace.offre.saveOffreAction"
            name="OffreForm"
            scope="request"
            validate="false"
            input="/pages/espace-recruteur/compte-recruteur/Offre.jsp">
            <forward name="continue" path="/pages/espace-recruteur/compte-
recruteur/consultation-offres.jsp"/>
    </action>
    <action path="/editOffre"
            type="jobplace.offre.editOffreAction"
            name="OffreForm"
            scope="request"
            validate="false"
            input="/pages/espace-recruteur/compte-recruteur/consulation-
offres.jsp">
            <forward name="continue" path="/pages/espace-recruteur/compte-
recruteur/Offre.jsp"/>
    </action>
    <action path="/supprimerOffre"
            type="jobplace.offre.SupprimerAction">
          <forward name="continue" path="/pages/espace-recruteur/compte-
recruteur/consultation-offres.jsp"/>
    </action>
    <action path="/ShowOffresEdit"
```

```
            type="jobplace.offre.showOffresEditAction">
          <forward name="continue" path="/pages/espace-recruteur/compte-
recruteur/consultation-offres.jsp"/>
    </action>
    <action path="/ShowOffres"
            type="jobplace.consultation.showOffresEmploiAction">
          <forward name="continue" path="/pages/espace-candidat/consultation-
offres.jsp"/>
    </action>
    <action path="/GetOffre"
            type="jobplace.consultation.getOffreAction">
          <forward name="continue" path="/pages/espace-candidat/showOffre.jsp"/>
    </action>
    <action path="/ShowCVs"
            type="jobplace.consultation.showCVsAction">
          <forward name="continue" path="/pages/espace-recruteur/consultation-
cvs.jsp"/>
    </action>
    <action path="/GetCV"
            type="jobplace.consultation.getCVAction">
          <forward name="continue" path="/pages/espace-recruteur/showCV.jsp"/>
    </action>
    <action path="/addToCaddy"
            type="jobplace.caddy.addToCaddyAction">
          <forward name="consultation-offres" path="/pages/espace-
candidat/consultation-offres.jsp"/>
          <forward name="showOffre" path="/pages/espace-candidat/showOffre.jsp"/>
          <forward name="consultation-cvs" path="/pages/espace-
recruteur/consultation-cvs.jsp"/>
          <forward name="showCV" path="/pages/espace-recruteur/showCV.jsp"/>
    </action>
    <action path="/supprimer"
            type="jobplace.caddy.SupprimerAction">
          <forward name="continue" path="/pages/caddy.jsp"/>
    </action>
    <!-- Process a user logoff -->
    <action    path="/Logoff"
               type="jobplace.logoff.LogoffAction">
      <forward name="success"             path="/pages/accueil.jsp"/>
    </action>
  </action-mappings>

</struts-config>
```

C.3: web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>

  <!-- Standard Action Servlet Configuration (with debugging) -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
```

```
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>validate</param-name>
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>

  <!-- Standard Action Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/do/*</url-pattern>
  </servlet-mapping>

  <!-- The Usual Welcome File List -->
  <welcome-file-list>
    <welcome-file>accueil.jsp</welcome-file>
  </welcome-file-list>

  <!-- Struts Tag Library Descriptors -->
  <taglib>
    <taglib-uri>/tags/struts-bean</taglib-uri>
    <taglib-location>/WEB-INF/taglibs/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/tags/struts-html</taglib-uri>
    <taglib-location>/WEB-INF/taglibs/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/tags/struts-logic</taglib-uri>
    <taglib-location>/WEB-INF/taglibs/struts-logic.tld</taglib-location>
  </taglib>

  <security-constraint>
      <display-name>Example Security Constraint</display-name>
      <web-resource-collection>
       <web-resource-name>Protected Area</web-resource-name>
        <!-- Define the context-relative URL(s) to be protected -->
       <url-pattern>/pages/espace-candidat/compte-candidat/*</url-pattern>
       <url-pattern>/do/editCV</url-pattern>
        <url-pattern>/do/saveCV</url-pattern>
        <url-pattern>/do/showLettres</url-pattern>
        <url-pattern>/do/editLettreMotivation</url-pattern>
        <url-pattern>/do/saveLettreMotivation</url-pattern>
        <url-pattern>/do/preparePostulation</url-pattern>
        <!-- If you list http methods, only those methods are protected -->
        <http-method>DELETE</http-method>
       <http-method>GET</http-method>
       <http-method>POST</http-method>
        <http-method>PUT</http-method>
      </web-resource-collection>
      <auth-constraint>
       <!-- Anyone with one of the listed roles may access this area -->
       <role-name>admin</role-name>
        <role-name>candidat</role-name>
      </auth-constraint>
    </security-constraint>
    <security-constraint>
      <display-name>Example Security Constraint</display-name>
      <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
       <!-- Define the context-relative URL(s) to be protected -->
      <url-pattern>/pages/espace-recruteur/compte-recruteur/*</url-pattern>
      <url-pattern>/do/saveOffre</url-pattern>
      <url-pattern>/do/editOffre</url-pattern>
      <url-pattern>/do/ShowOffresEdit</url-pattern>
```

```
      <!-- If you list http methods, only those methods are protected -->
      <http-method>DELETE</http-method>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      <http-method>PUT</http-method>
      </web-resource-collection>
      <auth-constraint>
       <!-- Anyone with one of the listed roles may access this area -->
       <role-name>admin</role-name>
       <role-name>recruteur</role-name>
      </auth-constraint>
    </security-constraint>

    <!-- Default login configuration uses form-based authentication -->
    <login-config>
      <auth-method>FORM</auth-method>
      <realm-name>Example Form-Based Authentication Area</realm-name>
      <form-login-config>
        <form-login-page>/pages/logon.jsp</form-login-page>
        <form-error-page>/pages/error.jsp</form-error-page>
      </form-login-config>
    </login-config>

</web-app>
```

C.4: extrait de poolman.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<poolman>

  <management-mode>local</management-mode>

  <datasource>

    <!-- ============================== -->
    <!-- Physical Connection Attributes -->
    <!-- ============================== -->

    <!-- Standard JDBC Driver info -->

    <dbname>USERS</dbname>
    <jndiName>jdbc/users</jndiName>
    <driver>org.gjt.mm.mysql.Driver</driver>
    <url>jdbc:mysql://localhost:3306/users</url>

    <username></username>
    <password></password>

    <minimumSize>0</minimumSize>
    <maximumSize>10</maximumSize>
    <connectionTimeout>600</connectionTimeout>
    <userTimeout>12</userTimeout>
    <shrinkBy>10</shrinkBy>

    <logFile>/var/applogs/poolman.log</logFile>
    <debugging>false</debugging>

    <!-- Query Cache Attributes-->

    <cacheEnabled>true</cacheEnabled>
    <cacheSize>20</cacheSize>
    <cacheRefreshInterval>120</cacheRefreshInterval>

  </datasource>

  <datasource>

    <!-- ============================== -->
```

```
    <!-- Physical Connection Attributes -->
    <!-- ============================== -->

    <!-- Standard JDBC Driver info -->

    <dbname>JOBPLACE</dbname>
    <jndiName>jdbc/jobplace</jndiName>
    <driver>org.gjt.mm.mysql.Driver</driver>
    <url>jdbc:mysql://localhost:3306/jobplace</url>

    <username></username>
    <password></password>

    <minimumSize>0</minimumSize>
    <maximumSize>10</maximumSize>
    <connectionTimeout>600</connectionTimeout>
    <userTimeout>12</userTimeout>
    <shrinkBy>10</shrinkBy>

    <logFile>/var/applogs/poolman.log</logFile>
    <debugging>false</debugging>

    <!-- Query Cache Attributes-->

    <cacheEnabled>true</cacheEnabled>
    <cacheSize>20</cacheSize>
    <cacheRefreshInterval>120</cacheRefreshInterval>

  </datasource>

</poolman>
```

C.5: extrait de ApplicationResoures_fr.properties

```
menubas=/pages/images/menubas-fr.gif
espace-candidat.gif=/pages/images/espace-candidat.gif
espace-recruteur.gif=/pages/images/espace-recruteur.gif
conseil.gif=/pages/images/conseil.gif
contact.gif=/pages/images/contact.gif
realisation=R&eacute;alis&eacute; par Yvette Feldmann
jobplace.adresse=jobplace, Fribourg

accueil.title=JobPlace
accueil.heading=<h1>Bienvenue &agrave; JobPlace</h1>
accueil.espace-recruteur=Espace recruteur
accueil.espace-candidat=Espace candidat

button.logon.submit=Login
button.submit=Envoyer
button.cancel=Annuler
button.reset=Effacer

caddy.title=Contenu du caddie
caddy.heading=<h1>Contenu du caddie</h1>
caddy.add=ajouter au caddie
caddy.show=Afficher contenu du caddie
caddy.delete=supprimer
caddy.empty=Le caddie est vide

candidat.candidat_titre=Titre
candidat.candidat_prenom=Pr&eacute;nom
candidat.candidat_nom=Nom
candidat.candidat_email=E-mail
candidat.candidat_adresse=Adresse
candidat.candidat_code_postal=Code postal
candidat.candidat_ville=Ville
candidat.candidat_pays=Pays
candidat.candidat_date_naissance=Date de naissance
candidat.candidat_phone=T&eacute;l&eacute;phone
```

```
candidat.candidat_portable=Portable

compte-candidat.title=JobPlace - compte-candidat
compte-candidat.heading=<h1>compte candidat</h1>
compte-candidat.cv.edit=Modifier
compte-candidat.cv.create=Cr&eacute;er
compte-candidat.cv.texte=le cv
compte-candidat.edit-inscription=Modifier le profil
compte-candidat.lettremotivation.edit=Lettres de motivation

compte-recruteur.title=JobPlace - compte-recruteur
compte-recruteur.heading=<h1>compte recruteur</h1>
compte-recruteur.offres=Offres d'emploi
compte-recruteur.edit-inscription=Modifier le profil
compte-recruteur.postulations=Consulter les postulations

connaissances_informatiques.titre=Connaissances informatiques
connaissances_informatiques.logiciels=Logiciels
connaissances_informatiques.langages=Langages
connaissances_informatiques.systemes_exploitations=Syst&egrave;mes d'exploitations
connaissances_informatiques.programmation=Programmation

connaissances_linguistiques.titre=Connaisances linguistiques
connaissances_linguistiques.langue_nom=Langue
connaissances_linguistiques.niveau=Niveau

conseil.title=Conseil
conseil.heading=<h1>Conseil</h1>

contact.title=Contact
contact.heading=<h1>Contact</h1>

consultation-cvs.title=CVth&egrave;ques
consultation-cvs.heading=<h1>CVth&egrave;ques</h1>
consultation-cvs.text=Choisissez un CV en cliquant sur le lien correspondant.

consultation-offres.title=Consultation des offres d'emploi
consultation-offres.heading=<h1>Consultation des offres d'emploi</h1>
consultation-offres.text=Choisissez une offre d'emploi en cliquant sur le lien
correspondant.

consultation-postulations.title=Consultation des postulations
consultation-postulations.heading=<h1>Consultation des postulations</h1>
consultation-postulations.show=afficher postulations

cv.nom=Nom
cv.prenom=Pr&eacute;nom
cv.adresse=Adresse
cv.ville=Ville

cv.title.create=Cr&eacute;er un nouveau CV
cv.title.edit=Modifier le CV
cv.header1.create=<h1>Nouveau CV</h1>
cv.header1.edit=<h1>Modifiez votre CV</h1>
cv.create=Veuillez remplir ce formulaire pour d&eacute;poser votre CV.
cv.edit=Veuillez actualiser les informations et sauvegarder les modifications.
cv.display=Voici les informations que vous avez entrées.
cv.note=<p><b>Note:</b> Les champs en gras <b>bold</b> sont requis.</p><p></p>
cv_id=CV id
cv_objectif=Objectif
cv.profil=Profil

espace-candidat.title=JobPlace - espace-candidat
espace-candidat.heading=<h1>espace-candidat</h1>
espace-candidat.registration=Inscription
espace-candidat.consultation-offres=Consultation offres-emploi
espace-candidat.compte-candidat=Compte personnel
```

C.6: log4j.properties

```
# Set root category priority to DEBUG and its only appender to A1.
log4j.rootCategory=DEBUG, stdout, R

# stdout is set to be a ConsoleAppender.
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%d [%t] %-5p (%F:%L) - %m%n

# Print only messages of priority INFO or above in the package jobplace
# log4j.category.jobplace=INFO

log4j.appender.R=org.apache.log4j.RollingFileAppender
# place of file
log4j.appender.R.File=logs/jobplace.log

log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

Annexe D: Code source

D.1: addToCaddyAction.java

```
package jobplace.caddy;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.Constants;
import jobplace.common.Caddy;
import jobplace.common.CaddyItem;
import jobplace.cv.CVForm;
import jobplace.offre.OffreEmploi;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Add CaddyItem to Caddy.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/13 $
 */
public class addToCaddyAction extends Action {

       // static variable for debugging and logging
       static Category cat = Category.getInstance(addToCaddyAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // method variables
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();

      ActionErrors errors = new ActionErrors();

      // get input_path
      String input_path = request.getParameter("input_path");

      if (input_path.equals("consultation-offres") ||
input_path.equals("showOffre")) {
             // we have to add offre
             OffreEmploi[] offres =
(OffreEmploi[])session.getAttribute(Constants.OFFRES_EMPLOI_ARRAY_KEY);
```

```
            if(offres == null) {
               errors.add(ActionErrors.GLOBAL_ERROR,
                   new ActionError("error.offres_emploi.missing"));
               // add a mapping to an error page.
            }

            if(offres != null) {
               Caddy caddy = (Caddy)session.getAttribute(Constants.CADDY_KEY);
               // if caddy exists for cv, create new
               if(caddy == null || caddy.getKind().equals("cv")) {
                  caddy = new Caddy();
                  caddy.setKind("offre");
               }

               // Check to see if Item already exists
               int offre_emploi_id =
Integer.parseInt(request.getParameter(Constants.OFFRE_EMPLOI_ID));
               OffreEmploi offre_emploi = null;

               if(!caddy.isCaddyItemPresent(offre_emploi_id)) {

                  // get offre_emploi corresponding to Id
                 for (int i=0;i<offres.length;i++) {
                    if (offre_emploi_id == offres[i].getOffre_emploi_id()) {
                           offre_emploi = offres[i];
                           caddy.addCaddyItem((CaddyItem)offre_emploi);
                           break;
                    }
                 }
               } else {
                   cat.info("Item " + offre_emploi_id + " is already in caddy.");

               }
               session.setAttribute(Constants.CADDY_KEY, caddy);
             }
      }

     if (input_path.equals("consultation-cvs") || input_path.equals("showCV")) {
            // we have to add cv
            CVForm[] cvforms =
(CVForm[])session.getAttribute(Constants.CVS_ARRAY_KEY);

            if(cvforms == null) {
               errors.add(ActionErrors.GLOBAL_ERROR,
                   new ActionError("error.cvform.missing"));
               // add a mapping to an error page.
            }

            if(cvforms != null) {
               Caddy caddy = (Caddy)session.getAttribute(Constants.CADDY_KEY);
               // if caddy exists for offre, create new
               if(caddy == null || caddy.getKind().equals("offre")) {
                  caddy = new Caddy();
                  caddy.setKind("cv");
               }
               // Check to see if Item already exists
               int cv_id = Integer.parseInt(request.getParameter(Constants.CV_ID));
               CVForm cvform = null;

               if(!caddy.isCaddyItemPresent(cv_id)) {

                  // get cvform corresponding to Id
                 for (int i=0;i<cvforms.length;i++) {
                    if (cv_id == cvforms[i].getCv_id()) {
                           cvform = cvforms[i];
                           caddy.addCaddyItem((CaddyItem)cvform);
                           break;
                    }
                 }
               } else {
```

```
                    cat.info("Item " + cv_id + " is already in caddy.");
                }
                session.setAttribute(Constants.CADDY_KEY, caddy);
              }
       }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (mapping.findForward("error"));
      }

      if (cat.isDebugEnabled()) {
       cat.debug("Input path = " + input_path + ", forward to it.");
      }
      return (mapping.findForward(input_path));

   }
}
```

D.2: jobplace/caddy/SupprimerAction.java

```
package jobplace.caddy;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.Constants;
import jobplace.common.Caddy;
import jobplace.common.CaddyItem;
import jobplace.offre.OffreEmploi;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Suppprimer CaddyItem of Caddy.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/13 $
 */
public class SupprimerAction extends Action {

   static Category cat = Category.getInstance(SupprimerAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // method variables
```

```
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();

      ActionErrors errors = new ActionErrors();

      Caddy caddy = (Caddy)session.getAttribute(Constants.CADDY_KEY);

      if(caddy == null) {
         errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.caddy.missing"));
      }

      if(caddy != null) {

         // get id
         int id = 0;
         String offre_emploi_id = request.getParameter(Constants.OFFRE_EMPLOI_ID);

         // is item offre_emploi
         if (offre_emploi_id != null) {
             id = Integer.parseInt(offre_emploi_id);
         } else {
             // item is cvform
             String cv_id = request.getParameter(Constants.CV_ID);
             id = Integer.parseInt(cv_id);
         }

         // get item
         CaddyItem item = caddy.getCaddyItem(id);
         // delete item
         caddy.deleteCaddyItem(item);

              cat.info("Item " + id + " has been deleted.");

         session.setAttribute(Constants.CADDY_KEY, caddy);
       }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (mapping.findForward("error"));
      }

      // Forward control
      if(cat.isDebugEnabled()) {
         cat.info("Forwarding to 'continue' page");
      }
      return mapping.findForward("continue");

   }
}
```

D.3: Caddy.java

```
package jobplace.common;

import java.io.Serializable;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class Caddy implements Serializable {
   private HashMap caddyItems = null;
   private int itemCount = 0;
   private String kind = null;

   public Caddy() {
      caddyItems = new HashMap();
```

```
   }

   public String getKind() {
       return kind;
   }
   public void setKind(String kind) {
       this.kind = kind;
   }

   public int getItemCount() {
      return itemCount;
   }

   private void setItemCount() {
      itemCount = caddyItems.size();
   }

   public Set entrySet() {
      return caddyItems.entrySet();
   }

   public Collection values() {
      return caddyItems.values();
   }

   public void addCaddyItem(CaddyItem caddyItem) {
      caddyItems.put(new Integer(caddyItem.getId()), caddyItem);
      setItemCount();
   }

   public void deleteCaddyItem(CaddyItem caddyItem) {
      Object oldItem = caddyItems.remove(new Integer(caddyItem.getId()));
      if(oldItem != null) {
         setItemCount();
      }
   }

   public CaddyItem getCaddyItem(int Id) {
      return (CaddyItem)caddyItems.get(new Integer(Id));
   }

   public boolean isCaddyItemPresent(int Id) {
      return caddyItems.containsKey(new Integer(Id));
   }

   // Needed for the struts-logic:iterate tag
   public CaddyItem[] getCaddyItems() {
      Collection collection = caddyItems.values();
      CaddyItem[] caddyItemArray = new CaddyItem[0];
      caddyItemArray = (CaddyItem[])collection.toArray(caddyItemArray);
      return caddyItemArray;
   }
}
```

D.4: CaddyItem.java

```
package jobplace.common;

public interface CaddyItem {

       public int getId();

}
```

D.5: Candidat.java

```
package jobplace.common;
```

```
import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;

public final class Candidat implements Serializable {

   private int candidat_id = 0;
   private String candidat_titre = null;
   private String candidat_nom = null;
   private String candidat_prenom = null;
   private String candidat_email = null;
   private String candidat_adresse = null;
   private String candidat_code_postal = null;
   private String candidat_ville = null;
   private String candidat_pays = null;
   private String candidat_date_naissance = null;
   private String candidat_phone = null;
   private String candidat_portable = null;

   public int getCandidat_id() {
      return candidat_id;
   }

   public void setCandidat_id(int candidat_id) {
      this.candidat_id = candidat_id;
   }

   public String getCandidat_titre() {
      return candidat_titre;
   }

   public void setCandidat_titre(String candidat_titre) {
      this.candidat_titre = candidat_titre;
   }

   public String getCandidat_prenom() {
      return candidat_prenom;
   }

   public void setCandidat_prenom(String candidat_prenom) {
      this.candidat_prenom = candidat_prenom;
   }

   public String getCandidat_nom() {
      return candidat_nom;
   }

   public void setCandidat_nom(String candidat_nom) {
      this.candidat_nom = candidat_nom;
   }

   public String getCandidat_email() {
      return candidat_email;
   }

   public void setCandidat_email(String candidat_email) {
      this.candidat_email = candidat_email;
   }

   public String getCandidat_adresse() {
      return candidat_adresse;
   }

   public void setCandidat_adresse(String candidat_adresse) {
      this.candidat_adresse = candidat_adresse;
   }
   public String getCandidat_ville() {
      return candidat_ville;
   }
```

```
    public void setCandidat_ville(String candidat_ville) {
        this.candidat_ville = candidat_ville;
    }

    public String getCandidat_pays() {
        return candidat_pays;
    }

    public void setCandidat_pays(String candidat_pays) {
        this.candidat_pays = candidat_pays;
    }

    public String getCandidat_code_postal() {
        return candidat_code_postal;
    }

    public void setCandidat_code_postal(String candidat_code_postal) {
        this.candidat_code_postal = candidat_code_postal;
    }

    public String getCandidat_date_naissance() {
        return candidat_date_naissance;
    }

    public void setCandidat_date_naissance(String candidat_date_naissance) {
        this.candidat_date_naissance = candidat_date_naissance;
    }

    public String getCandidat_phone() {
        return candidat_phone;
    }

    public void setCandidat_phone(String candidat_phone) {
        this.candidat_phone = candidat_phone;
    }

    public String getCandidat_portable() {
        return candidat_portable;
    }

    public void setCandidat_portable(String candidat_portable) {
        this.candidat_portable = candidat_portable;
    }

    public String toString() {
       StringBuffer sb = new StringBuffer("Candidat[");
       if (candidat_id > 0) {
          sb.append(", id=");
          sb.append(candidat_id);
       }
       if (candidat_nom != null) {
          sb.append(", nom=");
          sb.append(candidat_nom);
       }
       if (candidat_prenom != null) {
          sb.append(", prenom=");
          sb.append(candidat_prenom);
       }
       if(candidat_adresse != null) {
          sb.append(", adresse=");
          sb.append(candidat_adresse);
       }
       if(candidat_ville != null) {
          sb.append(", ville=");
          sb.append(candidat_ville);
       }
       sb.append("]");
       return sb.toString();
    }
}
```

D.6: Constants.java

```
package jobplace.common;

public final class Constants {

   // Session Context Keys
   public static final String CANDIDAT_KEY = "candidat_key";
   public static final String RECRUTEUR_KEY = "recruteur_key";
   public static final String CADDY_KEY = "caddy_key";
   public static final String OFFRES_EMPLOI_ARRAY_KEY = "offres_emploi_array_key";
   public static final String OFFRE_EMPLOI_KEY = "offre_emploi_key";
   public static final String CVFORM_KEY = "cvform_key";
   public static final String LETTRE_MOTIVATION_ARRAY_KEY =
"lettremotivation_array_key";
   public static final String LETTRE_MOTIVATION_KEY = "lettremotivation_key";
   public static final String POSTULATION_KEY = "postulation_key";
   public static final String POSTULATIONS_ARRAY_KEY = "postulations_array_key";
   public static final String CVS_ARRAY_KEY = "cvs_key";

   // Request Parameters
   public static final String OFFRE_EMPLOI_ID = "offre_emploi_id";
   public static final String LETTRE_MOTIVATION_ID = "lettre_motivation_id";
   public static final String CV_ID = "cv_id";
   public static final String POSTULATION_ID = "postulation_id";

}
```

D.7: Recruteur.java

```
package jobplace.common;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;

public final class Recruteur implements Serializable {

   private String societe_nom = null;
   private String societe_secteur_activite = null;
   private String societe_phone = null;
   private String societe_url = null;
   private String societe_description = null;
   private String contact_titre = null;
   private String contact_nom = null;
   private String contact_prenom = null;
   private String contact_email = null;
   private int fk_user_id = 0;

   public String getSociete_secteur_activite() {
       return societe_secteur_activite;
   }

   public void setSociete_secteur_activite(String societe_secteur_activite) {
       this.societe_secteur_activite = societe_secteur_activite;
   }

   public String getSociete_nom() {
       return societe_nom;
   }

   public void setSociete_nom(String societe_nom) {
       this.societe_nom = societe_nom;
   }

   public String getSociete_url() {
       return societe_url;
   }
```

```
public void setSociete_url(String societe_url) {
    this.societe_url = societe_url;
}
public String getSociete_phone() {
    return societe_phone;
}

public void setSociete_phone(String societe_phone) {
    this.societe_phone = societe_phone;
}

public String getSociete_description() {
    return societe_description;
}

public void setSociete_description(String societe_description) {
    this.societe_description = societe_description;
}
public String getContact_titre() {
    return contact_titre;
}

public void setContact_titre(String contact_titre) {
    this.contact_titre = contact_titre;
}

public String getContact_nom() {
    return contact_nom;
}

public void setContact_nom(String contact_nom) {
    this.contact_nom = contact_nom;
}

public String getContact_prenom() {
    return contact_prenom;
}

public void setContact_prenom(String contact_prenom) {
    this.contact_prenom = contact_prenom;
}
public String getContact_email() {
    return contact_email;
}

public void setContact_email(String contact_email) {
    this.contact_email = contact_email;
}

public int getFk_user_id() {
    return fk_user_id;
}

public void setFk_user_id(int fk_user_id) {
    this.fk_user_id = fk_user_id;
}

public String toString() {
   StringBuffer sb = new StringBuffer("Recruteur[");

   if (fk_user_id > 0) {
      sb.append(", fk_user_id=");
      sb.append(fk_user_id);
   }
   if (societe_nom != null) {
      sb.append(", nom=");
      sb.append(societe_nom);
   }
   if (societe_secteur_activite != null) {
      sb.append(", societe_secteur_activite=");
```

```
            sb.append(societe_secteur_activite);
         }
         if(societe_phone != null) {
            sb.append(", societe_phone=");
            sb.append(societe_phone);
         }
         if(societe_url != null) {
            sb.append(", adresse=");
            sb.append(societe_url);
         }
         if(societe_description != null) {
            sb.append(", societe_description=");
            sb.append(societe_description);
         }
         if(contact_nom != null) {
            sb.append(", contact_nom=");
            sb.append(contact_nom);
         }
         if(contact_prenom != null) {
            sb.append(", contact_prenom=");
            sb.append(contact_prenom);
         }
         if(contact_email != null) {
            sb.append(", contact_email=");
            sb.append(contact_email);
         }
         sb.append("]");
         return sb.toString();
      }
}
```

D.8: getCVAction.java

```
package jobplace.consultation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.cv.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show offre emploi.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class getCVAction extends Action {

       static Category cat = Category.getInstance(getCVAction.class.getName());
```

```
   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      CVForm[] cvforms = null;
      CVForm cvform = null;
      int cv_id = Integer.parseInt(request.getParameter("cv_id"));

      ActionErrors errors = new ActionErrors();

      if(cat.isDebugEnabled()) {
             cat.debug("GetCVAction:  Processing.");
      }

      cvforms = (CVForm[])session.getAttribute(Constants.CVS_ARRAY_KEY);

      if (cvforms==null) {
       cat.warn("cvforms is null");
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.cvforms.missing"));
      } else if (cv_id == 0) {
       cat.warn("cv_id is null");
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.cvform.missing"));
      } else {
       // get cv corresponding to cv_id
       for (int i=0;i<cvforms.length;i++) {
              if (cv_id == cvforms[i].getCv_id()) {
                     cvform = cvforms[i];
                     // set selected cvform into session
                     session.setAttribute(Constants.CVFORM_KEY,cvform);
                     return (mapping.findForward("continue"));
              }
       }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (mapping.findForward("error"));
      }

      cat.warn("No offre_emploi corresponds to the selected one.");
      errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.cvform.not_found"));

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
      }

      // Forward control
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.9: showCVsAction.java

```
package jobplace.consultation;

import java.io.IOException;
```

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.cv.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show CVs.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/15 $
 */
public class showCVsAction extends Action {

       static Category cat = Category.getInstance(showCVsAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
     HttpSession session = request.getSession();
      CVForm[] cvforms = null;

      if(cat.isDebugEnabled()) {
             cat.debug("showCVsAction:  Processing.");
      }
      try {
       cvforms = Statements.queryCVS(Commands.GET_CVS);
      } catch (SQLException sqle) {
             if (cat.isDebugEnabled()) {
                    cat.debug("Error during populating cvforms. " +
sqle.getMessage());
             }
             return mapping.findForward("error");
      }

      if (cvforms == null || cvforms.length==0) {
       cvforms = new CVForm[0];
         if (cat.isDebugEnabled()) {
             cat.debug("No cvs.");
       }
      }

      session.setAttribute(Constants.CVS_ARRAY_KEY, cvforms);

      // Forward control
      cat.info(" Forwarding to 'continue' page");
```

```
      return (mapping.findForward("continue"));
   }
}
```

D.10: getOffreAction.java

```
package jobplace.consultation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.offre.OffreEmploi;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show offre emploi.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class getOffreAction extends Action {

       static Category cat = Category.getInstance(getOffreAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      OffreEmploi[] offres = null;
      OffreEmploi offre_emploi = null;
      int offre_emploi_id =
Integer.parseInt(request.getParameter("offre_emploi_id"));

      ActionErrors errors = new ActionErrors();

      if(cat.isDebugEnabled()) {
             cat.debug("GetOffreAction:  Processing.");
      }

      offres =
(OffreEmploi[])session.getAttribute(Constants.OFFRES_EMPLOI_ARRAY_KEY);

      if (offres==null) {
       cat.warn("offres_emploi is null");
```

```
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.offres_emploi.missing"));
      } else if (offre_emploi_id == 0) {
       cat.warn("offre_emploi_id is null");
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.offre_emploi.missing"));
      } else {
       // get offre_emploi corresponding to offre_emploi_id
       for (int i=0;i<offres.length;i++) {
              if (offre_emploi_id == offres[i].getOffre_emploi_id()) {
                     offre_emploi = offres[i];
                     // set selected offre_emploi into session
                     session.setAttribute(Constants.OFFRE_EMPLOI_KEY,offre_emploi);
                     return (mapping.findForward("continue"));
              }
       }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (mapping.findForward("error"));
      }

      cat.warn("No offre_emploi corresponds to the selected one.");
      errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.offre_emploi.not_found"));

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
      }

      // Forward control
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.11: showOffresEmploiAction.java

```
package jobplace.consultation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.offre.OffreEmploi;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;
```

```
 /**
 * Show offres emploi.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class showOffresEmploiAction extends Action {

      static Category cat =
Category.getInstance(showOffresEmploiAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      OffreEmploi[] offres_emploi = null;

      if(cat.isDebugEnabled()) {
             cat.debug("ShowOffresEmploiAction:  Processing.");
      }
      try {
       // id == 0 to get all offres
       offres_emploi = Statements.queryOffres(Commands.GET_OFFRES_EMPLOI,0);
      } catch (SQLException sqle) {
             if (cat.isDebugEnabled()) {
                    cat.debug("Error during populating offres-emploi. " +
sqle.getMessage());
             }
             return mapping.findForward("error");
      }

      if (offres_emploi == null || offres_emploi.length==0) {
       offres_emploi = new OffreEmploi[0];
         if (cat.isDebugEnabled()) {
             cat.debug("No offres.");
       }
      }

      session.setAttribute(Constants.OFFRES_EMPLOI_ARRAY_KEY, offres_emploi);

      // Forward control
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.12: ConnaissancesInformatiques.java

```
package jobplace.cv;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;

public final class ConnaissancesInformatiques implements Serializable {

   private int informatique_id = 0;
   private String logiciels = null;
   private String langages = null;
   private String systemes_exploitations = null;
   private String programmation = null;

   public String getSystemes_exploitations() {
```

```
        return systemes_exploitations;
    }

    public void setSystemes_exploitations(String systemes_exploitations) {
        this.systemes_exploitations = systemes_exploitations;
    }

    public String getProgrammation() {
        return programmation;
    }

    public void setProgrammation(String programmation) {
        this.programmation = programmation;
    }

    public int getInformatique_id() {
        return informatique_id;
    }

    public void setInformatique_id(int informatique_id) {
        this.informatique_id = informatique_id;
    }

    public String getLogiciels() {
        return logiciels;
    }

    public void setLogiciels(String logiciels) {
        this.logiciels = logiciels;
    }

    public String getLangages() {
        return langages;
    }

    public void setLangages(String langages) {
        this.langages = langages;
    }

    public String toString() {
       StringBuffer sb = new StringBuffer("ConnaissancesInformatiques[");

       if (informatique_id > 0) {
          sb.append(", id=");
          sb.append(informatique_id);
       }
       if (logiciels != null) {
          sb.append(", logiciels=");
          sb.append(logiciels);
       }
       if (langages != null) {
          sb.append(", langages=");
          sb.append(langages);
       }
       if (langages != null) {
          sb.append(", systemes_exploitations=");
          sb.append(systemes_exploitations);
       }
       if (langages != null) {
          sb.append(", programmation=");
          sb.append(programmation);
       }
       sb.append("]");
       return sb.toString();
    }
}
```

D.13: ConnaissancesLinguistiques.java

```
package jobplace.cv;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;

public final class ConnaissancesLinguistiques implements Serializable {

   private int langue_id = 0;
   private String langue_nom = null;
   private String langue_niveau = null;

   public int getLangue_id() {
       return langue_id;
   }

   public void setLangue_id(int langue_id) {
       this.langue_id = langue_id;
   }

   public String getLangue_nom() {
       return langue_nom;
   }

   public void setLangue_nom(String langue_nom) {
       this.langue_nom = langue_nom;
   }

   public String getLangue_niveau() {
       return langue_niveau;
   }

   public void setLangue_niveau(String langue_niveau) {
       this.langue_niveau = langue_niveau;
   }

   public String toString() {
      StringBuffer sb = new StringBuffer("ConnaissancesLinguistiques[");

      if (langue_id > 0) {
         sb.append(", id=");
         sb.append(langue_id);
      }
      if (langue_nom != null) {
         sb.append(", langue_nom=");
         sb.append(langue_nom);
      }
      if (langue_niveau != null) {
         sb.append(", langue_niveau=");
         sb.append(langue_niveau);
      }
      sb.append("]");
      return sb.toString();
   }
}
```

D.14: CVForm.java

```
package jobplace.cv;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import java.util.HashMap;
```

```
import java.util.Map;

import jobplace.common.*;
 /**
 * FormBean for CV.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/02/14 $
 */
public class CVForm extends ActionForm implements jobplace.common.CaddyItem {

   private Candidat candidat = null;

   private int cv_id = 0;
   private String cv_date = null;
   private String cv_objectif = null;
   private Formation formation = null;
   private ConnaissancesLinguistiques langues = null;
   private ConnaissancesInformatiques informatiques = null;
   private ExperienceProfessionnelle profession = null;
   private HashMap mapping = new HashMap();  //mappings for link parameters


   private String action = "Create";

   // method for interface jobplace.common.CaddyItem
   public int getId() {
       return cv_id;
   }

   public String getAction() {
      return action;
   }

   public void setAction(String action) {
      this.action = action;
   }

   public Candidat getCandidat() {
       return candidat;
   }

   public void setCandidat(Candidat candidat) {
       this.candidat = candidat;
   }

   public int getCv_id() {
      return cv_id;
   }

   public void setCv_id(int cv_id) {
      this.cv_id = cv_id;
   }

   public String getCv_date() {
      return cv_date;
   }

   public void setCv_date(String cv_date) {
      this.cv_date = cv_date;
   }

   public String getCv_objectif() {
      return cv_objectif;
   }

   public void setCv_objectif(String cv_objectif) {
      this.cv_objectif = cv_objectif;
   }

   public Formation getFormation() {
```

```
      return formation;
   }

   public void setFormation(Formation formation) {
      this.formation = formation;
   }

   public ConnaissancesLinguistiques getLangues() {
      return langues;
   }

   public void setLangues(ConnaissancesLinguistiques langues) {
      this.langues = langues;
   }

   public ConnaissancesInformatiques getInformatiques() {
      return informatiques;
   }

   public void setInformatiques(ConnaissancesInformatiques informatiques) {
      this.informatiques = informatiques;
   }

   public ExperienceProfessionnelle getProfession() {
      return profession;
   }

   public void setProfession(ExperienceProfessionnelle profession) {
      this.profession = profession;
   }

   /**
    * The Mapping HashMap that is passed to the LinkTag in the form
    * tag library.  The HashMap is a collection of parameters that will
    * be used to make a query string and add it to the link.
    */
   public void setMapping() {
      mapping.put(Constants.CV_ID, new Integer(cv_id));
   }

   public Map getMapping() {
      return mapping;
   }

   public void reset(ActionMapping mapping, HttpServletRequest request) {
      action = "Create";
      cv_id = 0;
      cv_date = null;
      cv_objectif = null;
      this.candidat = new Candidat();
      this.formation = new Formation();
      this.langues = new ConnaissancesLinguistiques();
      this.informatiques = new ConnaissancesInformatiques();
      this.profession = new ExperienceProfessionnelle();
   }

   public ActionErrors validate(ActionMapping mapping,
                                HttpServletRequest request) {
      ActionErrors errors = new ActionErrors();
      return errors;
   }
}
```

D.15: editCVAction.java

```
package jobplace.cv;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
```

```
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Fill FormCV with CV from Database.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class editCVAction extends Action {

       static Category cat = Category.getInstance(editCVAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      String action = request.getParameter("action");
      String username = null;
      int candidat_id = 0;
      int result = 0;
      if(action == null) {
         action = "Create";
      }
      if(cat.isDebugEnabled()) {
             cat.debug("EditCVAction:  Processing " + action + " action");
      }

      CVForm cvform = (CVForm)form;

      Candidat candidat = (Candidat) session.getAttribute(Constants.CANDIDAT_KEY);

      if (candidat == null) {
             // Populate the candidat bean
             cat.info("Populate candidat bean.");

             // get username of logged user
             username = request.getRemoteUser();

             // check if user logged in
             if (username == null) {
                    cat.warn("User isn't logged!");
                    return mapping.findForward("error");
             }

             // get candidat_id
```

```
            try {
                  // get UserID in users
                  candidat_id =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error getting user id. " + sqle.getMessage());
                  }
            }
            if (candidat_id < 1) {
                  cat.warn("candidat_id is smaller than one.");
                  return mapping.findForward("error");
            }

            candidat = new Candidat();
            // Get Candidat Properties
            try {
                  candidat = (Candidat)
Statements.execQueryBean(candidat,Commands.GET_CANDIDAT_PROPERTIES,candidat_id);
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error getting Candidat properties. " +
sqle.getMessage());
                  }
            }

            if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean candidat: " +
candidat.toString());
            }

            candidat.setCandidat_id(candidat_id);
            session.setAttribute(Constants.CANDIDAT_KEY,candidat);
      }
      else {
            candidat_id = candidat.getCandidat_id();
      }

      if (action.equals("Create")) {
            //Check if user has got already a CV
            try {
                  result =
Statements.execQueryId("JOBPLACE",Commands.CHECK_CV,candidat_id);
            } catch (SQLException sqle) {
                  if (cat.isDebugEnabled()) {
                        cat.debug("Error during checking if cv already exists or
not. " + sqle.getMessage());
                  }
                  return mapping.findForward("error");
            }
            // If result is > 0, then cv exists already. -> change action to
"Edit"
            if (result > 0) {
                  action="Edit";
            }
            else if (result!=0) {
                  if(cat.isDebugEnabled()) {
                        cat.warn("DB Jobplace, table cv couldn't be checked.");
                  }
                  // Forward control to the specified success URI
                        return mapping.findForward("error");
                  }
      }
      if (action.equals("Edit")) {
        // populate cv

            Formation formation= new Formation();
            ConnaissancesLinguistiques langues = new ConnaissancesLinguistiques();
            ConnaissancesInformatiques informatiques = new
ConnaissancesInformatiques();
```

```
            ExperienceProfessionnelle profession = new
ExperienceProfessionnelle();
            int cv_id = cvform.getCv_id();
            try {
                  cvform = (CVForm)
Statements.execQueryBean(cvform,Commands.GET_CV_PROPERTIES,candidat_id);
                  formation = (Formation)
Statements.execQueryBean(formation,Commands.GET_FORMATION_PROPERTIES,cv_id);
                  if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean formation: " +
formation.toString());
                  }
                  langues = (ConnaissancesLinguistiques)
Statements.execQueryBean(langues,Commands.GET_CONNAISSANCES_LINGUISTIQUES_PROPERTIE
S,cv_id);
                  if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean langues: " +
langues.toString());
                  }
                  informatiques = (ConnaissancesInformatiques)
Statements.execQueryBean(informatiques,Commands.GET_CONNAISSANCES_INFORMATIQUES_PRO
PERTIES,cv_id);
                  if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean informatiques: " +
informatiques.toString());
                  }
                  profession = (ExperienceProfessionnelle)
Statements.execQueryBean(profession,Commands.GET_EXPERIENCE_PROFESSIONNELLE_PROPERT
IES,cv_id);
                  if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean profession: " +
profession.toString());
                  }
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error while populating cv. " +
sqle.getMessage());
                  }
                  return mapping.findForward("error");
            }

            cvform.setFormation(formation);
            cvform.setLangues(langues);
            cvform.setInformatiques(informatiques);
            cvform.setProfession(profession);
        }

        cvform.setAction(action);
        cvform.setCandidat(candidat);

        if("request".equals(mapping.getScope())) {
            request.setAttribute(mapping.getAttribute(), cvform);
        }

      // Forward control to the edit user registration page
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.16: ExperienceProfessionnelle.java

```
package jobplace.cv;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;
```

```
public final class ExperienceProfessionnelle implements Serializable {

   private int experience_id = 0;
   private String entreprise_nom = null;
   private String secteur_activite = null;
   private String poste = null;
   private String region = null;
   private String debut_travail = null;
   private String fin_travail = null;
   private String competences = null;

   public String getFin_travail() {
       return fin_travail;
   }

   public void setFin_travail(String fin_travail) {
       this.fin_travail = fin_travail;
   }

   public String getCompetences() {
       return competences;
   }

   public void setCompetences(String competences) {
       this.competences = competences;
   }

   public String getDebut_travail() {
       return debut_travail;
   }

   public void setDebut_travail(String debut_travail) {
       this.debut_travail = debut_travail;
   }

   public String getPoste() {
       return poste;
   }

   public void setPoste(String poste) {
       this.poste = poste;
   }

   public String getRegion() {
       return region;
   }

   public void setRegion(String region) {
       this.region = region;
   }

   public int getExperience_id() {
       return experience_id;
   }

   public void setExperience_id(int experience_id) {
       this.experience_id = experience_id;
   }

   public String getEntreprise_nom() {
       return entreprise_nom;
   }

   public void setEntreprise_nom(String entreprise_nom) {
       this.entreprise_nom = entreprise_nom;
   }

   public String getSecteur_activite() {
       return secteur_activite;
   }
```

```
   public void setSecteur_activite(String secteur_activite) {
       this.secteur_activite = secteur_activite;
   }

   public String toString() {
      StringBuffer sb = new StringBuffer("ExperienceProfessionnelle[");

      if (experience_id > 0) {
         sb.append(", id=");
         sb.append(experience_id);
      }
      if (entreprise_nom != null) {
         sb.append(", entreprise_nom=");
         sb.append(entreprise_nom);
      }
      if (secteur_activite != null) {
         sb.append(", secteur_activite=");
         sb.append(secteur_activite);
      }
      if (secteur_activite != null) {
         sb.append(", poste=");
         sb.append(poste);
      }
      if (secteur_activite != null) {
         sb.append(", region=");
         sb.append(region);
      }
      if (secteur_activite != null) {
         sb.append(", debut_travail=");
         sb.append(debut_travail);
      }
      if (secteur_activite != null) {
         sb.append(", fin_travail=");
         sb.append(fin_travail);
      }
      if (secteur_activite != null) {
         sb.append(", competences=");
         sb.append(competences);
      }
      sb.append("]");
      return sb.toString();
   }
}
```

D.17: Formation.java

```
package jobplace.cv;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;

public final class Formation implements Serializable {

   private int formation_id = 0;
   private String etablissement_nom = null;
   private String formation_duree = null;
   private String diplome_nom = null;
   private String diplome_lieu = null;

   public int getFormation_id() {
       return formation_id;
   }

   public void setFormation_id(int formation_id) {
       this.formation_id = formation_id;
   }
```

```
   public String getEtablissement_nom() {
       return etablissement_nom;
   }

   public void setEtablissement_nom(String etablissement_nom) {
       this.etablissement_nom = etablissement_nom;
   }

   public String getFormation_duree() {
       return formation_duree;
   }

   public void setFormation_duree(String formation_duree) {
       this.formation_duree = formation_duree;
   }

   public String getDiplome_nom() {
       return diplome_nom;
   }

   public void setDiplome_nom(String diplome_nom) {
       this.diplome_nom = diplome_nom;
   }

   public String getDiplome_lieu() {
       return diplome_lieu;
   }

   public void setDiplome_lieu(String diplome_lieu) {
       this.diplome_lieu = diplome_lieu;
   }

   public String toString() {
      StringBuffer sb = new StringBuffer("Formation[");

      if (formation_id > 0) {
         sb.append(", id=");
         sb.append(formation_id);
      }
      if (etablissement_nom != null) {
         sb.append(", etablissement_nom=");
         sb.append(etablissement_nom);
      }
      if (formation_duree != null) {
         sb.append(", formation_duree=");
         sb.append(formation_duree);
      }
      if(diplome_nom != null) {
         sb.append(", diplome_nom=");
         sb.append(diplome_nom);
      }
      if(diplome_lieu != null) {
         sb.append(", diplome_lieu=");
         sb.append(diplome_lieu);
      }
      sb.append("]");
      return sb.toString();
   }
}
```

D.18: saveCVAction.java

```
package jobplace.cv;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.util.Date;
import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.postulation.PostulationForm;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Save CV in Database.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/02/14 $
 */
public class saveCVAction extends Action {

       static Category cat = Category.getInstance(saveCVAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // method variables
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();

      ActionErrors errors = new ActionErrors();

      String action = request.getParameter("action");
      String forward = "compte-candidat";
      CVForm cvform = (CVForm) form;

      if(action == null) {
         action = "Create";
      }

      cat.info("SaveCVAction:  Processing " + action + " action");

      // Was this transaction cancelled?
      if(isCancelled(request)) {
         cat.info(" Transaction '" + action + "' was cancelled");

         if(mapping.getAttribute() != null) {
            session.removeAttribute(mapping.getAttribute());
         }
         // if canceled go to compte-candidat
         return mapping.findForward("compte-candidat");
      }

      // obtain the input values from the form
      Candidat candidat = cvform.getCandidat();
      Formation formation = cvform.getFormation();
      ConnaissancesLinguistiques langues = cvform.getLangues();
      ConnaissancesInformatiques informatiques = cvform.getInformatiques();
      ExperienceProfessionnelle profession = cvform.getProfession();
```

```
      int candidat_id=candidat.getCandidat_id();
      String cv_date = (new Date()).toString();
      String cv_object = cvform.getCv_objectif();
      int cv_id=0;
      int result=0;

      if (action.equals("Create")) {
       // save candidat
       try {
              result = Statements.updateCandidat(Commands.CANDIDAT_UPDATE,candidat);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, candidat couldn't be inserted.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error insert candidat. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       // save cv
       try {
              result =
Statements.updateCV(Commands.CV_INSERT,cv_date,cv_object,candidat_id);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, cv couldn't be inserted.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error insert cv. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       // get cv_id from cv
       try {
              cv_id =
Statements.execQueryId("JOBPLACE",Commands.GET_CV_ID,candidat_id);
       } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error getting cv_id. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       if (cv_id < 1) {
              cat.warn("CV_id is smaller than one.");
              return mapping.findForward("error");
       }

       // set cv_id
       cvform.setCv_id(cv_id);

       // insert formation
       try {
              result =
Statements.updateFormation(Commands.FORMATION_INSERT,formation,cv_id);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, formation couldn't be
inserted.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
```

```
            }
        } catch (SQLException sqle) {
            if(cat.isDebugEnabled()) {
                cat.debug("Error insert formation. " + sqle.getMessage());
            }
            return mapping.findForward("error");
      }
      // insert connaissances_linguistiques
      try {
            result =
Statements.updateLangues(Commands.CONNAISSANCES_LINGUISTIQUES_INSERT,langues,cv_id)
;
            if (result==0) {
                if(cat.isDebugEnabled()) {
                    cat.debug("DB Jobplace, langues couldn't be inserted.");
                }
                // Forward control to the specified success URI
                return mapping.findForward("error");
            }
        } catch (SQLException sqle) {
            if(cat.isDebugEnabled()) {
                cat.debug("Error insert langues. " + sqle.getMessage());
            }
            return mapping.findForward("error");
      }
      // insert connaissances_informatiques
      try {
            result =
Statements.updateInformatiques(Commands.CONNAISSANCES_INFORMATIQUES_INSERT,informat
iques,cv_id);
            if (result==0) {
                if(cat.isDebugEnabled()) {
                    cat.debug("DB Jobplace, informatiques couldn't be
inserted.");
                }
                // Forward control to the specified success URI
                return mapping.findForward("error");
            }
        } catch (SQLException sqle) {
            if(cat.isDebugEnabled()) {
                cat.debug("Error insert informatiques. " + sqle.getMessage());
            }
            return mapping.findForward("error");
      }
      // insert experience_professionnelle
      try {
            result =
Statements.updateProfession(Commands.EXPERIENCE_PROFESSIONNELLE_INSERT,profession,c
v_id);
            if (result==0) {
                if(cat.isDebugEnabled()) {
                    cat.debug("DB Jobplace, profession couldn't be
inserted.");
                }
                // Forward control to the specified success URI
                return mapping.findForward("error");
            }
        } catch (SQLException sqle) {
            if(cat.isDebugEnabled()) {
                cat.debug("Error insert profession. " + sqle.getMessage());
            }
            return mapping.findForward("error");
      }
     } else {
      // action==Edit

      // update cv
      try {
            result =
Statements.updateCV(Commands.CV_UPDATE,cv_date,cv_object,candidat_id);
```

```
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                           cat.debug("DB Jobplace, cv couldn't be updated.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error update cv. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       // update formation
       // get id from formation
       int id = formation.getFormation_id();

       try {
              // if id<1 then formation doesn't exist -> insert
              if (id < 1) {
                     cat.debug("edit: formation_id is smaller than one, insert.");
                     result =
Statements.updateFormation(Commands.FORMATION_INSERT,formation,cv_id);
              } else {
                     result =
Statements.updateFormation(Commands.FORMATION_UPDATE,formation,id);
              }
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                           cat.debug("DB Jobplace, formation couldn't be updated.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error update formation. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       // update langues
       // get id from langues
       id = langues.getLangue_id();
       try {
              // if id<1 then langues doesn't exist -> insert
              if (id < 1) {
                     cat.debug("edit: langue_id is smaller than one, insert.");
                     result =
Statements.updateLangues(Commands.CONNAISSANCES_LINGUISTIQUES_INSERT,langues,cv_id)
;
              } else {
                     result =
Statements.updateLangues(Commands.CONNAISSANCES_LINGUISTIQUES_UPDATE,langues,id);
              }
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                           cat.debug("DB Jobplace, langues couldn't be updated.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error update langues. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       // update informatiques
       // get id from informatiques
       id = informatiques.getInformatique_id();
```

```
      try {
            // if id<1 then informatiques doesn't exist -> insert
            if (id < 1) {
                  cat.debug("edit: informatique_id is smaller than one, insert.");
                  result =
Statements.updateInformatiques(Commands.CONNAISSANCES_INFORMATIQUES_INSERT,informat
iques,cv_id);
            } else {
                  result =
Statements.updateInformatiques(Commands.CONNAISSANCES_INFORMATIQUES_UPDATE,informat
iques,id);
            }
            if (result==0) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("DB Jobplace, informatiques couldn't be
updated.");
                  }
                  // Forward control to the specified success URI
                  return mapping.findForward("error");
            }
       } catch (SQLException sqle) {
            if(cat.isDebugEnabled()) {
                  cat.debug("Error update informatiques. " + sqle.getMessage());
            }
            return mapping.findForward("error");
      }
      // update profession
      // get id from profession
      id = profession.getExperience_id();

      try {
            // if id<1 then profession doesn't exist -> insert
            if (id < 1) {
                  cat.debug("edit: experience_id is smaller than one, insert.");
                  result =
Statements.updateProfession(Commands.EXPERIENCE_PROFESSIONNELLE_INSERT,profession,c
v_id);
            } else {
                  result =
Statements.updateProfession(Commands.EXPERIENCE_PROFESSIONNELLE_UPDATE,profession,i
d);
            }
            if (result==0) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("DB Jobplace, profession couldn't be
updated.");
                  }
                  // Forward control to the specified success URI
                  return mapping.findForward("error");
            }
       } catch (SQLException sqle) {
            if(cat.isDebugEnabled()) {
                  cat.debug("Error update profession. " + sqle.getMessage());
            }
            return mapping.findForward("error");
      }
     }

      // update session-candidat bean
      session.setAttribute(Constants.CANDIDAT_KEY,candidat);

      PostulationForm postulationform = (PostulationForm)
session.getAttribute(Constants.POSTULATION_KEY);
      if (postulationform != null) {
            if ("Step-CreateCV".equals(postulationform.getAction())) {
                  postulationform.setCvform(cvform);
                  postulationform.setAction("Step-CVSet");
                  session.setAttribute(Constants.POSTULATION_KEY,postulationform);
                  forward = "preparePostulation";
```

```
            }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
            saveErrors(request, errors);
            return (new ActionForward(mapping.getInput()));
      }

      // Remove the obsolete form bean
      if(mapping.getAttribute() != null) {
         if("request".equals(mapping.getScope())) {
            request.removeAttribute(mapping.getAttribute());
         } else {
            session.removeAttribute(mapping.getAttribute());
         }
      }

      // Forward control to the specified success URI
      return mapping.findForward(forward);
   }
}
```

D.19: editInscriptionAction.java

```
package jobplace.inscription;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.Candidat;
import jobplace.common.Recruteur;
import jobplace.common.Constants;
import jobplace.sql.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Fill InscriptionForm with Inscription-Details from Database.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/02/14 $
 */
public class editInscriptionAction extends Action {

   // static variable for debugging and logging
   static Category cat =
Category.getInstance(editInscriptionAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
```

```
   throws IOException, ServletException {



      // method variables
      HttpSession session = request.getSession();
      // values of action: Edit or Create
      String action = request.getParameter("action");
      // values of user_role: Candidat or Recruteur
      String user_role = request.getParameter("user_role");
      String username = null;
      int userid = 0;
      Candidat candidat = null;
      Recruteur recruteur = null;

InscriptionForm inscriptionform = (InscriptionForm)form;

      ActionErrors errors = new ActionErrors();

      if(action == null) {
         action = "Create";
      }
      if(user_role == null){
 // todo
      }

      if(cat.isDebugEnabled()) {
             cat.debug("EditInscriptionAction:  Processing " + action + " action");
      }

      // get username of logged user
      username = request.getRemoteUser();

      if (action.equals("Create")) {
       // check if user already logged in
       if (username != null) {
             action = "Edit";
       } else if (user_role.equals("candidat")) {
             candidat = new Candidat();
             inscriptionform.setCandidat(candidat);
       } else {
             recruteur = new Recruteur();
             inscriptionform.setRecruteur(recruteur);
       }
      }

      // action == Edit
      if (action.equals("Edit")) {

             cat.info("User is a " + user_role + ".");

             // is user Candidat?
             if (user_role.equals("candidat")) {

                    // get session's candidat object
                    candidat = (Candidat)
session.getAttribute(Constants.CANDIDAT_KEY);

                    // if object doesn't exist yet, create candidat object
                    if (candidat == null) {

                           // Populate the Candidat
                           cat.info("Create candidat bean and populate it.");

                           try {
                                  // get UserID in users
                                  userid =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
                           } catch (SQLException sqle) {
```

```
                                    if(cat.isDebugEnabled()) {
                                          cat.debug("Error getting user id. " +
sqle.getMessage());
                                    }
                              }

                              candidat = new Candidat();

                              // Get Candidat Properties
                              try {
                                    candidat = (Candidat)
Statements.execQueryBean(candidat,Commands.GET_CANDIDAT_PROPERTIES,userid);
                              } catch (SQLException sqle) {
                                    if(cat.isDebugEnabled()) {
                                          cat.debug("Error getting Candidat
properties. " + sqle.getMessage());
                                    }
                              }

                              // show candidat's data
                              if(cat.isDebugEnabled()) {
                                    cat.debug("Populated bean candidat: " +
candidat.toString());
                              }

                              // set candidat_id
                              candidat.setCandidat_id(userid);

                              session.setAttribute(Constants.CANDIDAT_KEY,candidat);
                        }

                        inscriptionform.setCandidat(candidat);

                  }
                  // user is Recruteur
                  else {
                        // get session's candidat object
                        recruteur = (Recruteur)
session.getAttribute(Constants.RECRUTEUR_KEY);

                        // if object doesn't exist yet, create candidat object
                        if (recruteur == null) {

                              // Populate the recruteur
                              cat.info("Create recruteur bean and populate it.");

                              try {
                                    // get UserID in users
                                    userid =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
                              } catch (SQLException sqle) {
                                    if(cat.isDebugEnabled()) {
                                          cat.debug("Error getting user id. " +
sqle.getMessage());
                                    }
                              }

                              recruteur = new Recruteur();

                              // Get recruteur Properties
                              try {
                                    recruteur = (Recruteur)
Statements.execQueryBean(recruteur,Commands.GET_RECRUTEUR_PROPERTIES,userid);
                              } catch (SQLException sqle) {
                                    if(cat.isDebugEnabled()) {
                                          cat.debug("Error getting recruteur
properties. " + sqle.getMessage());
                                    }
                              }
```

```
                        // show recruteur's data
                        if(cat.isDebugEnabled()) {
                              cat.debug("Populated bean recruteur: " +
recruteur.toString());
                        }

                        // set recruteur_id
                        recruteur.setFk_user_id(userid);

                        session.setAttribute(Constants.RECRUTEUR_KEY,recruteur);
                  }

                  inscriptionform.setRecruteur(recruteur);
            }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
            saveErrors(request, errors);
            return (mapping.findForward("error"));
      }

      // set values for inscriptionform
      inscriptionform.setAction(action);
      inscriptionform.setUser_role(user_role);

      if("request".equals(mapping.getScope())) {
                  request.setAttribute(mapping.getAttribute(), inscriptionform);
      }

     // Forward control to the edit user registration page
     cat.info(" Forwarding to 'continue' page");

     return (mapping.findForward("continue"));
   }
}
```

D.20: InscriptionForm.java

```
package jobplace.inscription;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import jobplace.common.Candidat;
import jobplace.common.Recruteur;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * FormBean for Inscription.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/02/13 $
 */
public class InscriptionForm extends ActionForm {
   private int user_id = 0;
   private String user_name = null;
   private String user_pass = null;
   private String user_role = null;
   private Candidat candidat = null;
   private Recruteur recruteur = null;

   private String action = "Create";
```

```
static Category cat = Category.getInstance(InscriptionForm.class.getName());

public int getUser_id() {
    return user_id;
}

public void setUser_id(int user_id) {
    this.user_id = user_id;
}

public String getUser_name() {
   return user_name;
}

public void setUser_name(String user_name) {
   this.user_name = user_name;
}

public String getUser_pass() {
   return user_pass;
}

public void setUser_pass(String user_pass) {
   this.user_pass = user_pass;
}

public String getUser_role() {
   return user_role;
}

public void setUser_role(String user_role) {
   this.user_role = user_role;
}

public Candidat getCandidat() {
   return candidat;
}

public void setCandidat(Candidat candidat) {
   this.candidat = candidat;
}

public Recruteur getRecruteur() {
   return recruteur;
}

public void setRecruteur(Recruteur recruteur) {
   this.recruteur = recruteur;
}

public String getAction() {
   return action;
}

public void setAction(String action) {
   this.action = action;
}

public void reset(ActionMapping mapping, HttpServletRequest request) {
   user_id = 0;
   user_name = null;
   user_pass = null;
   user_role = null;
   this.candidat = new Candidat();
   this.recruteur = new Recruteur();
   action = "Create";
}

public ActionErrors validate(ActionMapping mapping,
                          HttpServletRequest request) {
```

```
      ActionErrors errors = new ActionErrors();
      if((user_name == null) || (user_name.length() < 1)) {
         errors.add("user_name",
             new ActionError("error.user_name.required"));
      }
      if((user_pass == null) || (user_pass.length() < 1)) {
         errors.add("user_pass",
             new ActionError("error.user_pass.required"));
      }
      if((user_role == null) || (user_role.length() < 1)) {
        cat.error("Put data in InscriptionForm: user_role required!");
         errors.add("user_role",
             new ActionError("error.user_role.required"));
      }

      return errors;
   }
}
```

D.21: saveInscriptionAction.java

```
package jobplace.inscription;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.Constants;
import jobplace.common.Candidat;
import jobplace.common.Recruteur;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Save Candidat in Database Users and Jobplace.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class saveInscriptionAction extends Action {

       // static variable for debugging and logging
       static Category cat =
Category.getInstance(saveInscriptionAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // method variables
      HttpSession session = request.getSession();
```

```
      ServletContext servletContext = servlet.getServletContext();

      ActionErrors errors = new ActionErrors();
      // action == Create or Edit
      String action = request.getParameter("action");
      // user_role == candidat or recruteur
      String user_role = request.getParameter("user_role");
      // get user bean if exist
      Candidat session_candidat = (Candidat)
session.getAttribute(Constants.CANDIDAT_KEY);
      Recruteur session_recruteur = (Recruteur)
session.getAttribute(Constants.RECRUTEUR_KEY);
      int result=0;
      InscriptionForm inscriptionform = (InscriptionForm) form;
      // obtain the user properties from the form
      String user_name = inscriptionform.getUser_name();
      String user_pass = inscriptionform.getUser_pass();
      // forward uri
      String forward = "cancel";


      if(action == null) {
         action = "Create";
      }

      // Was this transaction cancelled?
      if(isCancelled(request)) {
         cat.info(" Transaction '" + action + "' was cancelled");

         if(mapping.getAttribute() != null) {
            session.removeAttribute(mapping.getAttribute());
         }

         return(mapping.findForward("cancel"));
      }

      if (user_role == null) {
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.user_role.required"));
       cat.error("SaveInscriptionAction - Error: User_role is null!!");
      }
      // user_role == candidat or recruteur
      else {
       cat.info("SaveInscriptionAction:  Processing " + action + " action for " +
user_role);


       if (action.equals("Create")) {
              // if action==create save data first in db users

              // check if user_name already exists
              try {
                     result =
Statements.execQueryUsername("USERS",Commands.CHECK_USERNAME,user_name);
              } catch (SQLException sqle) {
                     if (cat.isDebugEnabled()) {
                            cat.debug("Error during checking if user_name already
exists or not. " + sqle.getMessage());
                     }
                     return mapping.findForward("error");
              }
              // If result is > 0, then user_name exists already.
              if (result > 0) {
                     errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.user_name.already_exists"));
                     cat.info("SaveInscriptionAction: User_name already exists!");
              }
              else if (result==0) {
                     // save data in table users from users
                     try {
```

```
                        result =
Statements.updateUsers(Commands.USER_INSERT,user_name,user_pass);

                        if (result!=0) {
                              result =
Statements.updateUsers(Commands.USER_ROLES_INSERT,user_name,user_role);
                        }
                        else {
                              // error updating users
                              if(cat.isDebugEnabled()) {
                                    cat.warn("DB Users, Table users couldn't be
updated.");
                              }
                              // Forward control to the specified success URI
                              return mapping.findForward("error");
                        }
                        // error updating user_roles
                        if (result==0) {
                              if(cat.isDebugEnabled()) {
                                    cat.warn("DB Users, Table user_roles
couldn't be updated.");
                              }
                              // Forward control to the specified success URI
                              return mapping.findForward("error");
                        }
                  } catch (SQLException sqle) {
                        if(cat.isDebugEnabled()) {
                              cat.debug("Error while saving user in users. " +
sqle.getMessage());
                        }
                        return mapping.findForward("error");
                  }
            } // END: save in users

            // save user in jobplace
            try {
                  result =
Statements.updateUserJobplace(Commands.USER_JOBPLACE_INSERT,user_name,user_pass,use
r_role);
                  if (result==0) {
                        if(cat.isDebugEnabled()) {
                              cat.warn("DB Jobplace, Table users couldn't be
updated.");
                        }
                        // Forward control to the specified success URI
                        return mapping.findForward("error");
                  }
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error while saving user in jobplace. " +
sqle.getMessage());
                  }
                  return mapping.findForward("error");
            }

      } // END: action==create
     }
     // Report any errors we have discovered back to the original form
     if(!errors.empty()) {
      saveErrors(request, errors);
        return (new ActionForward(mapping.getInput()));
     }

     // save candidat
     if (user_role.equals("candidat")) {

      // set candidat bean
      Candidat candidat = inscriptionform.getCandidat();

      // get userid
```

```
      int userid=0;

      if (action.equals("Create")) {
            try {
              // get UserID in users
              userid =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,user_name);
            } catch (SQLException sqle) {
             if(cat.isDebugEnabled()) {
                   cat.debug("SaveInscription: Error getting user id. " +
sqle.getMessage());
             }
             return mapping.findForward("error");
            }
            if (userid < 1) {
             cat.warn("User_id is smaller than one.");
             return mapping.findForward("error");
            }
      } else {
            userid = session_candidat.getCandidat_id();
      }

        candidat.setCandidat_id(userid);

      // save candidat in table candidat
      if (action.equals("Create")) {

            try {
                  result =
Statements.updateCandidat(Commands.CANDIDAT_INSERT,candidat);
                  if (result==0) {
                        if(cat.isDebugEnabled()) {
                              cat.debug("DB Jobplace, candidat couldn't be
inserted.");
                        }
                        // Forward control to the specified success URI
                        return mapping.findForward("error");
                  }
               } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("SaveInscription: Error insert candidat. " +
sqle.getMessage());
                  }
                  return mapping.findForward("error");
            }
      }
      else {
            //action==edit
            try {
                  result =
Statements.updateCandidat(Commands.CANDIDAT_UPDATE,candidat);
                  if (result==0) {
                        if(cat.isDebugEnabled()) {
                              cat.debug("DB Jobplace, Table candidat couldn't be
updated.");
                        }
                        // Forward control to the specified success URI
                        return mapping.findForward("error");
                  }
               } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("SaveInscription: Error update candidat. " +
sqle.getMessage());
                  }
                  return mapping.findForward("error");
            }
      }
      session.setAttribute(Constants.CANDIDAT_KEY,candidat);
      forward = "continue-candidat";
     }
```

```
      else {
       //save recruteur

       // set recruteur bean
       Recruteur recruteur = inscriptionform.getRecruteur();

       // get userid as foreign key
       int userid=0;

       if (action.equals("Create")) {
             // if Create get user_id from db
             try {
               // get UserID in users
               userid =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,user_name);
             } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                    cat.debug("SaveInscription: Error getting user id. " +
sqle.getMessage());
              }
              return mapping.findForward("error");
            }
            if (userid < 1) {
              cat.warn("User_id is smaller than one.");
              return mapping.findForward("error");
            }
       } else {
             // action == edit
             userid = session_recruteur.getFk_user_id();
       }

       recruteur.setFk_user_id(userid);

       // save recruteur in table recruteur
       if (action.equals("Create")) {

             try {
                    result =
Statements.updateRecruteur(Commands.RECRUTEUR_INSERT,recruteur);
                    if (result==0) {
                          if(cat.isDebugEnabled()) {
                                cat.debug("DB Jobplace, recruteur couldn't be
inserted.");
                          }
                          // Forward control to the specified success URI
                          return mapping.findForward("error");
                    }
               } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                          cat.debug("SaveInscription: Error insert recruteur. " +
sqle.getMessage());
                    }
                    return mapping.findForward("error");
             }
       }
       else {
             //action==edit
             try {
                    result =
Statements.updateRecruteur(Commands.RECRUTEUR_UPDATE,recruteur);
                    if (result==0) {
                          if(cat.isDebugEnabled()) {
                                cat.debug("DB Jobplace, Table candidat couldn't be
updated.");
                          }
                          // Forward control to the specified success URI
                          return mapping.findForward("error");
                    }
               } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
```

```
                        cat.debug("SaveInscription: Error update candidat. " +
sqle.getMessage());
                    }
                    return mapping.findForward("error");
             }

      }
      session.setAttribute(Constants.RECRUTEUR_KEY,recruteur);
      forward = "continue-recruteur";

     } // END: save candidat or recruteur

     // Remove the obsolete form bean
     if(mapping.getAttribute() != null) {
        if("request".equals(mapping.getScope())) {
           request.removeAttribute(mapping.getAttribute());
        } else {
           session.removeAttribute(mapping.getAttribute());
        }
     }

     // Forward control to the specified success URI
     return mapping.findForward(forward);
  }
}
```

D.22: editLettreMotivationAction.java

```
package jobplace.lettremotivation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Fill form with lettre motivation
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/11 $
 */
public class editLettreMotivationAction extends Action {

      static Category cat =
Category.getInstance(editLettreMotivationAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
```

```
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      String action = request.getParameter("action");
      String username = null;

      int candidat_id = 0;
      int result = 0;

      ActionErrors errors = new ActionErrors();

      if(action == null) {
         action = "Create";
      }
      if(cat.isDebugEnabled()) {
            cat.debug("EditLettreMotivationAction:  Processing " + action + "
action");
      }

      LettreMotivationForm lettreform = (LettreMotivationForm)form;

      Candidat candidat = (Candidat) session.getAttribute(Constants.CANDIDAT_KEY);

      if (candidat == null) {
            // Populate the candidat bean
            cat.info("Populate candidat bean.");

            // get username of logged user
            username = request.getRemoteUser();

            // check if user logged in
            if (username == null) {
                  cat.warn("User isn't logged!");
                  return mapping.findForward("error");
            }

            // get candidat_id
            try {
                  // get UserID in users
                  candidat_id =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error getting user id. " + sqle.getMessage());
                  }
            }
            if (candidat_id < 1) {
                  cat.warn("candidat_id is smaller than one.");
                  return mapping.findForward("error");
            }

            candidat = new Candidat();
            // Get Candidat Properties
            try {
                  candidat = (Candidat)
Statements.execQueryBean(candidat,Commands.GET_CANDIDAT_PROPERTIES,candidat_id);
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error getting Candidat properties. " +
sqle.getMessage());
                  }
            }

            if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean candidat: " +
candidat.toString());
            }
```

```
              candidat.setCandidat_id(candidat_id);
              session.setAttribute(Constants.CANDIDAT_KEY,candidat);
       }
       else {
              candidat_id = candidat.getCandidat_id();
       }

       if (action.equals("Create")) {
              lettreform.setFk_candidat_id(candidat_id);
       }
       if (action.equals("Edit")) {
              int lettre_motivation_id =
Integer.parseInt(request.getParameter("lettre_motivation_id"));
              // get lettre motivatation
                LettreMotivationForm[] lettres =
(LettreMotivationForm[])session.getAttribute(Constants.LETTRE_MOTIVATION_ARRAY_KEY)
;

              if (lettres==null) {
                     cat.warn("lettres_motivation is null");
                     errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.lettres_motivation.missing"));
              } else if (lettre_motivation_id == 0) {
                     cat.warn("lettre_motivation_id is null");
                     errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.lettre_motivation.missing"));
              } else {
                     // get lettre_motivation corresponding to lettre_motivation_id
                     if (cat.isDebugEnabled()) {
                            cat.debug("lettre_motivation_id == " +
lettre_motivation_id);
                     }
                     for (int i=0;i<lettres.length;i++) {
                            if (lettre_motivation_id ==
lettres[i].getLettre_motivation_id()) {
                                   if (cat.isDebugEnabled()) {
                                          cat.debug("lettre-motivation found");
                                   }
                                   lettreform = lettres[i];
                            }
                     }
              }
       }

       lettreform.setAction(action);

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
              saveErrors(request, errors);
              return (new ActionForward(mapping.getInput()));
      }

        if("request".equals(mapping.getScope())) {
              request.setAttribute(mapping.getAttribute(), lettreform);
        }

      // Forward control to the edit user registration page
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.23: showLettresMotivationAction.java

```
package jobplace.lettremotivation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
```

```
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show lettres motivation.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/11 $
 */
public class showLettresMotivationAction extends Action {

      static Category cat =
Category.getInstance(showLettresMotivationAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      LettreMotivationForm[] lettres = null;
      int candidat_id = 0;
      String username = null;

      if(cat.isDebugEnabled()) {
             cat.debug("ShowLettresMotivationAction:  Processing.");
      }

      // get candidat_id
      Candidat candidat = (Candidat) session.getAttribute(Constants.CANDIDAT_KEY);

      if (candidat == null) {
             // Populate the candidat bean
             cat.info("Populate candidat bean.");

             // get username of logged user
             username = request.getRemoteUser();

             // check if user logged in
             if (username == null) {
                    cat.warn("User isn't logged!");
                    return mapping.findForward("error");
             }

             // get candidat_id
             try {
                    // get UserID in users
```

```
                    candidat_id =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
              } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                          cat.debug("Error getting user id. " + sqle.getMessage());
                    }
              }
              if (candidat_id < 1) {
                    cat.warn("candidat_id is smaller than one.");
                    return mapping.findForward("error");
              }

              candidat = new Candidat();
              // Get Candidat Properties
              try {
                    candidat = (Candidat)
Statements.execQueryBean(candidat,Commands.GET_CANDIDAT_PROPERTIES,candidat_id);
              } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                          cat.debug("Error getting Candidat properties. " +
sqle.getMessage());
                    }
              }

              if(cat.isDebugEnabled()) {
                          cat.debug("Populated bean candidat: " +
candidat.toString());
              }

              candidat.setCandidat_id(candidat_id);
              session.setAttribute(Constants.CANDIDAT_KEY,candidat);
       }
       else {
              candidat_id = candidat.getCandidat_id();
       } // END: get candidat_id


      // get lettres for candidat_id
      try {
       lettres =
Statements.queryLettres(Commands.GET_LETTRES_MOTIVATION,candidat_id);
      } catch (SQLException sqle) {
              if (cat.isDebugEnabled()) {
                    cat.debug("Error during populating lettres-motivation. " +
sqle.getMessage());
              }
              return mapping.findForward("error");
      }
      if (lettres == null || lettres.length==0) {
       lettres = new LettreMotivationForm[0];
         if (cat.isDebugEnabled()) {
              cat.debug("No lettres de motivation.");
       }
      } else {
       // show only the first ten characters of lettre_motivation_texte
       for (int i=0;i<lettres.length;i++) {
              String tmp = lettres[i].getLettre_motivation_texte();
              if (tmp.length() > 19) {
                    lettres[i].setLettre_motivation_texte_tmp(tmp.substring(0,19));
              } else lettres[i].setLettre_motivation_texte_tmp(tmp);
       }
      }
      session.setAttribute(Constants.LETTRE_MOTIVATION_ARRAY_KEY, lettres);

      // Forward control to the edit user registration page
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.24: saveLettreMotivationAction.java

```
package jobplace.lettremotivation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.util.Date;
import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.*;
import jobplace.postulation.PostulationForm;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Save LettreMotivation in Database.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/12 $
 */
public class saveLettreMotivationAction extends Action {

       static Category cat =
Category.getInstance(saveLettreMotivationAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // method variables
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();
      String forward="continue";

      ActionErrors errors = new ActionErrors();

      String action = request.getParameter("action");

      LettreMotivationForm lettreform = (LettreMotivationForm) form;

      if(action == null) {
         action = "Create";
      }

      cat.info("SaveLettreMotivationAction:  Processing " + action + " action");

      // Was this transaction cancelled?
      if(isCancelled(request)) {
         cat.info(" Transaction '" + action + "' was cancelled");

         if(mapping.getAttribute() != null) {
            session.removeAttribute(mapping.getAttribute());
```

```
         }
         // if canceled go to compte-candidat
         return mapping.findForward(forward);
      }

      // obtain the input values from the form
      String lettre_motivation_texte = lettreform.getLettre_motivation_texte();
      int fk_candidat_id = lettreform.getFk_candidat_id();
      int result=0;

      if (action.equals("Create")) {
       // save lettre-motivation
       try {
              result =
Statements.updateLettreMotivation(Commands.LETTRE_MOTIVATION_INSERT,lettre_motivati
on_texte,fk_candidat_id);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, lettre_motivation couldn't be
inserted.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error insert lettre_motivation. " +
sqle.getMessage());
              }
              return mapping.findForward("error");
       }
      } else {
       // action==Edit

       int lettre_motivation_id = lettreform.getLettre_motivation_id();

       // update lettre-motivation
       try {
              result =
Statements.updateLettreMotivation(Commands.LETTRE_MOTIVATION_UPDATE,lettre_motivati
on_texte,lettre_motivation_id);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, lettre_motivation couldn't be
updated.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error update lettre_motivation. " +
sqle.getMessage());
              }
              return mapping.findForward("error");
       }
      }

       // if createLettreMotivation was invoked by postulation go to postulation
       PostulationForm postulationform = (PostulationForm)
session.getAttribute(Constants.POSTULATION_KEY);
       if (postulationform != null) {
              if ("Step-CreateLettreMotivation".equals(postulationform.getAction()))
{
                     postulationform.setLettreform(lettreform);
                     postulationform.setAction("Step-LettreMotivationSet");
                     session.setAttribute(Constants.POSTULATION_KEY,postulationform);
                     forward = "preparePostulation";
              }
       }
```

```
       // Report any errors we have discovered back to the original form
       if(!errors.empty()) {
              saveErrors(request, errors);
              return (new ActionForward(mapping.getInput()));
       }

      // Remove the obsolete form bean
      if(mapping.getAttribute() != null) {
         if("request".equals(mapping.getScope())) {
            request.removeAttribute(mapping.getAttribute());
            if (cat.isDebugEnabled()) {
             cat.debug("remove obsolete form bean");
           }
         } else {
            session.removeAttribute(mapping.getAttribute());
         }
      }

      // Forward control to the specified success URI
      cat.info(" Forwarding to '" + forward + "' page");
      return mapping.findForward(forward);
   }
}
```

D.25: LettreMotivationForm.java

```
package jobplace.lettremotivation;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import jobplace.common.*;

import java.util.HashMap;
import java.util.Map;
 /**
 * FormBean for LettreMotivation.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/11 $
 */
public class LettreMotivationForm extends ActionForm {

   private String action = null;

   private int lettre_motivation_id = 0;
   private String lettre_motivation_texte = null;
   // variable to save first characters of lettre_motivation_texte
   private String lettre_motivation_texte_tmp = null;
   private int fk_candidat_id = 0;
   private HashMap mapping = new HashMap();  //mappings for link parameters

   public LettreMotivationForm() {
   }

   public String getAction() {
       return action;
   }
   public void setAction(String action) {
       this.action = action;
   }

   public int getLettre_motivation_id() {
      return lettre_motivation_id;
```

```
   }

   public void setLettre_motivation_id(int lettre_motivation_id) {
      this.lettre_motivation_id = lettre_motivation_id;
   }

   public String getLettre_motivation_texte() {
      return lettre_motivation_texte;
   }

   public void setLettre_motivation_texte(String lettre_motivation_texte) {
      this.lettre_motivation_texte = lettre_motivation_texte;
   }

   public String getLettre_motivation_texte_tmp() {
      return lettre_motivation_texte_tmp;
   }

   public void setLettre_motivation_texte_tmp(String lettre_motivation_texte_tmp) {
      this.lettre_motivation_texte_tmp = lettre_motivation_texte_tmp;
   }

   public int getFk_candidat_id() {
      return fk_candidat_id;
   }

   public void setFk_candidat_id(int fk_candidat_id) {
      this.fk_candidat_id = fk_candidat_id;
   }

   /**
    * The Mapping HashMap that is passed to the LinkTag in the form
    * tag library.  The HashMap is a collection of parameters that will
    * be used to make a query string and add it to the link.
    */
   public void setMapping() {
      mapping.put(Constants.LETTRE_MOTIVATION_ID, new
Integer(lettre_motivation_id));
   }

   public Map getMapping() {
      return mapping;
   }

   public String toString() {
      StringBuffer sb = new StringBuffer("Category[lettre_motivation_id=");
      sb.append(lettre_motivation_id);
      sb.append(", lettre_motivation_texte=");
      sb.append(lettre_motivation_texte);
      sb.append(", fk_candidat_id");
      sb.append(fk_candidat_id);
      sb.append("]");
      return sb.toString();
   }

   public void reset(ActionMapping mapping, HttpServletRequest request) {
      action = "Create";
   }

   public ActionErrors validate(ActionMapping mapping,
                                HttpServletRequest request) {
      ActionErrors errors = new ActionErrors();

      return errors;
   }
}
```

D.26: jobplace/lettremotivation/SupprimerAction.java

```
package jobplace.lettremotivation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.util.Date;
import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.postulation.PostulationForm;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Delete selected lettre in table lettre_motivation.
 * Receives lettre_motivation_id as parameter.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/17 $
 */
public class SupprimerAction extends Action {

       static Category cat = Category.getInstance(SupprimerAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // method variables
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();
      int result = 0;
      ActionErrors errors = new ActionErrors();

      String lettre_motivation_id = request.getParameter("lettre_motivation_id");

      if (lettre_motivation_id == null) {
       cat.warn("error.lettre_motivation.missing");
       errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.lettre_motivation.missing"));
      }
      else {
       // delete lettre_motivation

       cat.info("SupprimerAction:  Processing deleting lettre_motivation");

       try {
              result = Statements.deleteQuery(Commands.LETTRE_MOTIVATION_DELETE,
Integer.parseInt(lettre_motivation_id));
       } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
```

```
                    cat.debug("Error deleting lettre_motivation. " +
sqle.getMessage());
             }
             errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.lettre_motivation.deleting"));
       }
       if (result < 1) {
             if(cat.isDebugEnabled()) {
                    cat.debug("Couldn't delete lettre_motivation. ");
             }
             errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.lettre_motivation.deleting"));
       }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
             saveErrors(request, errors);
             return (new ActionForward(mapping.getInput()));
      }

      // Forward control to the specified success URI
      return mapping.findForward("continue");
   }
}
```

D.27: LogoffAction.java

```
package jobplace.logoff;

import java.io.IOException;
import java.util.Hashtable;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;
import jobplace.common.Constants;
import jobplace.common.User;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Logoff: invalidate session.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/02/14 $
 */
public class LogoffAction extends Action {

   static Category cat = Category.getInstance(LogoffAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // Extract attributes we will need
      HttpSession session = request.getSession();
      //User user = (User)session.getAttribute(Constants.USER_KEY);
```

```
       String user = request.getRemoteUser();

      // Process this user logoff
      if (user != null) {
         cat.info("Logoff: User '" + user +
               "' logged off in session " + session.getId());

      } else {
         cat.info("LogoffActon: User logged off in session " +
               session.getId());
      }
      //session.removeAttribute(Constants.USER_KEY);
      session.invalidate();

      // Forward control to the specified success URI
      return (mapping.findForward("success"));
   }
}
```

D.28: editOffreAction.java

```
package jobplace.offre;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Prepares the form of Offre.jsp. If action==Create then create a new instance of
 * OffreEmploi. If action==Edit then get OffreEmploi from the array with the
 * corresponding offre_emploi_id.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class editOffreAction extends Action {

       static Category cat = Category.getInstance(editOffreAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      String action = request.getParameter("action");
```

```
      int offre_emploi_id = 0;
      String username = null;
      OffreEmploi[] offres = null;
      int fk_user_id = 0;
      int result = 0;

      if(action == null) {
         action = "Create";
      }
      if(cat.isDebugEnabled()) {
             cat.debug("EditOffreAction:  Processing " + action + " action");
      }

      OffreEmploi offre_emploi = (OffreEmploi)form;
      ActionErrors errors = new ActionErrors();

      Recruteur recruteur = (Recruteur)
session.getAttribute(Constants.RECRUTEUR_KEY);

      if (recruteur == null) {
             // Populate the recruteur bean
             cat.info("Populate recruteur bean.");

             // get username of logged user
             username = request.getRemoteUser();

             // check if user logged in
             if (username == null) {
                    cat.warn("User isn't logged!");
                    return mapping.findForward("error");
             }

             // get fk_user_id
             try {
                    // get UserID in users
                    fk_user_id =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
             } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                           cat.debug("Error getting user id. " + sqle.getMessage());
                    }
             }
             if (fk_user_id < 1) {
                    cat.warn("recruteur_id is smaller than one.");
                    return mapping.findForward("error");
             }

             recruteur = new Recruteur();
             // Get recruteur Properties
             try {
                    recruteur = (Recruteur)
Statements.execQueryBean(recruteur,Commands.GET_RECRUTEUR_PROPERTIES,fk_user_id);
             } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                           cat.debug("Error getting recruteur properties. " +
sqle.getMessage());
                    }
             }

             if(cat.isDebugEnabled()) {
                           cat.debug("Populated bean recruteur: " +
recruteur.toString());
             }

             recruteur.setFk_user_id(fk_user_id);
             session.setAttribute(Constants.RECRUTEUR_KEY,recruteur);
       }
       else {
             fk_user_id = recruteur.getFk_user_id();
       }
```

```
      if (action.equals("Create")) {
            offre_emploi = new OffreEmploi();
            offre_emploi.setFk_recruteur_id(fk_user_id);
      }
      if (action.equals("Edit")) {

           offres =
(OffreEmploi[])session.getAttribute(Constants.OFFRES_EMPLOI_ARRAY_KEY);

           if (offres==null) {
            cat.warn("offres_emploi is null");
            errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.offres_emploi.missing"));
           } else {
            offre_emploi_id =
Integer.parseInt(request.getParameter("offre_emploi_id"));
            if (offre_emploi_id == 0) {
                  cat.warn("offre_emploi_id is null");
                  errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.offre_emploi.missing"));
            } else {
                  // get offre_emploi corresponding to offre_emploi_id
                  for (int i=0;i<offres.length;i++) {
                        if (offre_emploi_id == offres[i].getOffre_emploi_id()) {
                              offre_emploi = offres[i];
                              break;
                        }

                  }
            }
           }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (mapping.findForward("error"));
      }

      offre_emploi.setAction(action);

      if("request".equals(mapping.getScope())) {
            request.setAttribute(mapping.getAttribute(), offre_emploi);
      }

      // Forward control to the edit user registration page
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.29: showOffresEditAction.java

```
package jobplace.offre;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
```

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show offres emploi to edit.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/05 $
 */
public class showOffresEditAction extends Action {

      static Category cat =
Category.getInstance(showOffresEditAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      OffreEmploi[] offres_emploi = null;
      int fk_user_id = 0;
      String username = null;

      if(cat.isDebugEnabled()) {
             cat.debug("ShowOffresEmploiAction:  Processing.");
      }

      Recruteur recruteur = (Recruteur)
session.getAttribute(Constants.RECRUTEUR_KEY);

      if (recruteur == null) {
             // Populate the recruteur bean
             cat.info("Populate recruteur bean.");

             // get username of logged user
             username = request.getRemoteUser();

             // check if user logged in
             if (username == null) {
                    cat.warn("User isn't logged!");
                    return mapping.findForward("error");
             }

             // get fk_user_id
             try {
                    // get UserID in users
                    fk_user_id =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
             } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                           cat.debug("Error getting user id. " + sqle.getMessage());
                    }
             }
             if (fk_user_id < 1) {
                    cat.warn("recruteur_id is smaller than one.");
                    return mapping.findForward("error");
             }
```

```
            recruteur = new Recruteur();
            // Get recruteur Properties
            try {
                  recruteur = (Recruteur)
Statements.execQueryBean(recruteur,Commands.GET_RECRUTEUR_PROPERTIES,fk_user_id);
            } catch (SQLException sqle) {
                  if(cat.isDebugEnabled()) {
                        cat.debug("Error getting recruteur properties. " +
sqle.getMessage());
                  }
            }

            if(cat.isDebugEnabled()) {
                        cat.debug("Populated bean recruteur: " +
recruteur.toString());
            }

            recruteur.setFk_user_id(fk_user_id);
            session.setAttribute(Constants.RECRUTEUR_KEY,recruteur);
      }
      else {
            fk_user_id = recruteur.getFk_user_id();
      }

      // get offres for fk_user_id
      try {
       offres_emploi =
Statements.queryOffres(Commands.GET_OFFRES_EMPLOI_RECRUTEUR,fk_user_id);
      } catch (SQLException sqle) {
            if (cat.isDebugEnabled()) {
                  cat.debug("Error during populating offres-emploi. " +
sqle.getMessage());
            }
            return mapping.findForward("error");
      }

      if (offres_emploi == null || offres_emploi.length==0) {
       offres_emploi = new OffreEmploi[0];
         if (cat.isDebugEnabled()) {
            cat.debug("No offres.");
       }
      }

      session.setAttribute(Constants.OFFRES_EMPLOI_ARRAY_KEY, offres_emploi);

      // Forward control
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.30: saveOffreAction.java

```
package jobplace.offre;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
```

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.util.Date;
import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.postulation.PostulationForm;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Save Offre in Database.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/17 $
 */
public class saveOffreAction extends Action {

       static Category cat = Category.getInstance(saveOffreAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {

      // method variables
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();

      ActionErrors errors = new ActionErrors();

      String action = request.getParameter("action");
      OffreEmploi offre = (OffreEmploi) form;

      if(action == null) {
         action = "Create";
      }

      cat.info("SaveOffreAction:  Processing " + action + " action");

      // Was this transaction cancelled?
      if(isCancelled(request)) {
         cat.info(" Transaction '" + action + "' was cancelled");

         if(mapping.getAttribute() != null) {
            session.removeAttribute(mapping.getAttribute());
         }
         // if canceled go to compte-candidat
         return mapping.findForward("continue");
      }

      // obtain the input values from the form
      String offre_emploi_poste = offre.getOffre_emploi_poste();
      String offre_emploi_description_poste =
offre.getOffre_emploi_description_poste();
      String offre_emploi_qualites_requises =
offre.getOffre_emploi_qualites_requises();
      String offre_emploi_connaissances_techniques =
offre.getOffre_emploi_connaissances_techniques();
      String offre_emploi_region = offre.getOffre_emploi_region();

      // set date
      String offre_emploi_date = (new Date()).toString();
```

```
      int result=0;

      if (action.equals("Create")) {
       // save offre
       int fk_recruteur_id = offre.getFk_recruteur_id();
       try {
              result =
Statements.updateOffre(Commands.OFFRE_INSERT,offre_emploi_date, offre_emploi_poste,
offre_emploi_description_poste, offre_emploi_qualites_requises,
offre_emploi_connaissances_techniques, offre_emploi_region, fk_recruteur_id);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, offre_emploi couldn't be
inserted.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error insert offre_emploi. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
      } else {
       // action==Edit

       int offre_emploi_id = offre.getOffre_emploi_id();
       // update offre_emploi
       try {
              result =
Statements.updateOffre(Commands.OFFRE_UPDATE,offre_emploi_date, offre_emploi_poste,
offre_emploi_description_poste, offre_emploi_qualites_requises,
offre_emploi_connaissances_techniques, offre_emploi_region, offre_emploi_id);
              if (result==0) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("DB Jobplace, offre_emploi couldn't be
updated.");
                     }
                     // Forward control to the specified success URI
                     return mapping.findForward("error");
              }
        } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error update offre_emploi. " + sqle.getMessage());
              }
              return mapping.findForward("error");
       }
      }

       // Report any errors we have discovered back to the original form
       if(!errors.empty()) {
              saveErrors(request, errors);
              return (new ActionForward(mapping.getInput()));
       }

      // Remove the obsolete form bean
      if(mapping.getAttribute() != null) {
         if("request".equals(mapping.getScope())) {
            request.removeAttribute(mapping.getAttribute());
         } else {
            session.removeAttribute(mapping.getAttribute());
         }
      }

      // Forward control to the specified success URI
      return mapping.findForward("continue");
   }
}
```

D.31: OffreEmploi.java

```
package jobplace.offre;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

import jobplace.common.Constants;

public class OffreEmploi extends ActionForm implements jobplace.common.CaddyItem {
   private int offre_emploi_id = 0;
   private String offre_emploi_date = null;
   private String offre_emploi_poste = null;
   private String offre_emploi_description_poste = null;
   private String offre_emploi_qualites_requises = null;
   private String offre_emploi_connaissances_techniques = null;
   private String offre_emploi_region = null;
   private int fk_recruteur_id = 0;
   private String action = "Create";
   private HashMap mapping = new HashMap();  //mappings for link parameters

   public OffreEmploi() {
   }

   public String getAction() {
       return action;
   }

   public void setAction(String action) {
       this.action = action;
   }

   // method for interface CaddyItem
   public int getId() {
       return offre_emploi_id;
   }

   public int getOffre_emploi_id() {
      return offre_emploi_id;
   }

   public void setOffre_emploi_id(int offre_emploi_id) {
      this.offre_emploi_id = offre_emploi_id;
   }

   public String getOffre_emploi_date() {
      return offre_emploi_date;
   }

   public void setOffre_emploi_date(String offre_emploi_date) {
      this.offre_emploi_date = offre_emploi_date;
   }

   public String getOffre_emploi_poste() {
      return offre_emploi_poste;
   }

   public void setOffre_emploi_poste(String offre_emploi_poste) {
      this.offre_emploi_poste = offre_emploi_poste;
   }

   public String getOffre_emploi_description_poste() {
      return offre_emploi_description_poste;
```

```
   }

   public void setOffre_emploi_description_poste(String
offre_emploi_description_poste) {
      this.offre_emploi_description_poste = offre_emploi_description_poste;
   }
   public String getOffre_emploi_qualites_requises() {
      return offre_emploi_qualites_requises;
   }

   public void setOffre_emploi_qualites_requises(String
offre_emploi_qualites_requises) {
      this.offre_emploi_qualites_requises = offre_emploi_qualites_requises;
   }
   public String getOffre_emploi_connaissances_techniques() {
      return offre_emploi_connaissances_techniques;
   }

   public void setOffre_emploi_connaissances_techniques(String
offre_emploi_connaissances_techniques) {
      this.offre_emploi_connaissances_techniques =
offre_emploi_connaissances_techniques;
   }

   public String getOffre_emploi_region() {
      return offre_emploi_region;
   }

   public void setOffre_emploi_region(String offre_emploi_region) {
      this.offre_emploi_region = offre_emploi_region;
   }
   public int getFk_recruteur_id() {
      return fk_recruteur_id;
   }

   public void setFk_recruteur_id(int fk_recruteur_id) {
      this.fk_recruteur_id = fk_recruteur_id;
   }

   /**
    * The Mapping HashMap that is passed to the LinkTag in the form
    * tag library.  The HashMap is a collection of parameters that will
    * be used to make a query string and add it to the link.
    */
   public void setMapping() {
      mapping.put(Constants.OFFRE_EMPLOI_ID, new Integer(offre_emploi_id));
   }

   public Map getMapping() {
      return mapping;
   }

   public void reset(ActionMapping mapping, HttpServletRequest request) {
      action = "Create";
      offre_emploi_id = 0;
      offre_emploi_date = null;
      offre_emploi_poste = null;
      offre_emploi_description_poste = null;
      offre_emploi_qualites_requises = null;
      offre_emploi_connaissances_techniques = null;
      offre_emploi_region = null;
      fk_recruteur_id = 0;
   }

   public ActionErrors validate(ActionMapping mapping,
                            HttpServletRequest request) {
      ActionErrors errors = new ActionErrors();
      return errors;
   }
```

```
   public String toString() {
      StringBuffer sb = new StringBuffer("Category[offre_emploi_id=");
      sb.append(offre_emploi_id);
      sb.append(", offre_emploi_date=");
      sb.append(offre_emploi_date);
      sb.append(", offre_emploi_poste=");
      sb.append(offre_emploi_poste);
      sb.append(", offre_emploi_description_poste=");
      sb.append(offre_emploi_description_poste);
      sb.append(", offre_emploi_qualites_requises=");
      sb.append(offre_emploi_qualites_requises);
      sb.append(", offre_emploi_connaissances_techniques=");
      sb.append(offre_emploi_connaissances_techniques);
      sb.append(", offre_emploi_region");
      sb.append(offre_emploi_region);
      sb.append(", fk_recruteur_id");
      sb.append(fk_recruteur_id);
      sb.append("]");
      return sb.toString();
   }
}
```

D.32: jobplace/offre/SupprimerAction.java

```
package jobplace.offre;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.util.Date;
import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.postulation.PostulationForm;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Delete selected offre in table offre_emploi.
 * Receives offre_emploi_id as parameter.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/17 $
 */
public class SupprimerAction extends Action {

       static Category cat = Category.getInstance(SupprimerAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {
```

```
      // method variables
      HttpSession session = request.getSession();
      ServletContext servletContext = servlet.getServletContext();
      int result = 0;
      ActionErrors errors = new ActionErrors();

      String offre_emploi_id = request.getParameter("offre_emploi_id");

      if (offre_emploi_id == null) {
       cat.warn("error.offre_emploi.missing");
       errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.offre_emploi.missing"));
      }
      else {
       // delete offre

       cat.info("SupprimerAction:  Processing deleting offre_emploi");

       try {
              result = Statements.deleteQuery(Commands.OFFRE_EMPLOI_DELETE,
Integer.parseInt(offre_emploi_id));
       } catch (SQLException sqle) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Error deleting offre. " + sqle.getMessage());
              }
              errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.offre_emploi.deleting"));
       }
       if (result < 1) {
              if(cat.isDebugEnabled()) {
                     cat.debug("Couldn't delete offre.");
              }
              errors.add(ActionErrors.GLOBAL_ERROR,new
ActionError("error.offre_emploi.deleting"));
       }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
              saveErrors(request, errors);
              return (new ActionForward(mapping.getInput()));
      }

      // Forward control to the specified success URI
      return mapping.findForward("continue");
   }
}
```

D.33: getPostulationAction.java

```
package jobplace.postulation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;
```

```
import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.postulation.PostulationForm;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show postulations.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/19 $
 */
public class getPostulationAction extends Action {

      static Category cat =
Category.getInstance(getPostulationAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      PostulationForm[] postulations = null;
      PostulationForm postulation = null;
      int postulation_id =
Integer.parseInt(request.getParameter("postulation_id"));

      ActionErrors errors = new ActionErrors();

      if(cat.isDebugEnabled()) {
             cat.debug("GetPostulationAction:  Processing.");
      }

      postulations =
(PostulationForm[])session.getAttribute(Constants.POSTULATIONS_ARRAY_KEY);

      if (postulations==null) {
       cat.warn("postulations is null");
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.postulations.missing"));
      } else if (postulation_id == 0) {
       cat.warn("postulation_id is null");
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.postulation.missing"));
      } else {
       // get postulation corresponding to postulation_id
       for (int i=0;i<postulations.length;i++) {
             if (postulation_id == postulations[i].getPostulation_id()) {
                   postulation = postulations[i];
                   // set selected postulation into session
                   session.setAttribute(Constants.POSTULATION_KEY,postulation);
                   return (mapping.findForward("continue"));
             }
       }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (mapping.findForward("error"));
      }

      cat.warn("No postulation corresponds to the selected one.");
```

```
      errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.postulation.not_found"));

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
      }

      // Forward control
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.34: PostulationAction.java

```
package jobplace.postulation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.offre.OffreEmploi;

import java.util.Date;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Takes the postulationform with its components. Inserts the postulation in the
 * database with the request Commands.POSTULATION_INSERT
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/17 $
 */
public class PostulationAction extends Action {

       static Category cat =
Category.getInstance(PostulationAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      int lettreform_id = 0;
      int offre_emploi_id = 0;
```

```
      int cvform_id = 0;
      int result = 0;

      ActionErrors errors = new ActionErrors();

      PostulationForm postulation = (PostulationForm) form;

      // Was this transaction cancelled?
      if(isCancelled(request)) {
         cat.info(" Transaction 'postulation' was cancelled");

         if(mapping.getAttribute() != null) {
            session.removeAttribute(mapping.getAttribute());
         }

         return(mapping.findForward("cancel"));
      }

      if (postulation == null) {
       cat.info("No postulation is selected");
       errors.add(ActionErrors.GLOBAL_ERROR, new
ActionError("error.postulation.missing"));
      } else {
      // save postulation
       if(cat.isDebugEnabled()) {
              cat.debug("PostulationAction:  Processing");
       }

         lettreform_id = postulation.getLettreform().getLettre_motivation_id();
       offre_emploi_id = postulation.getOffre_emploi().getOffre_emploi_id();
       cvform_id = postulation.getCvform().getCv_id();
       String postulation_date = (new Date()).toString();

       try {
              result =
Statements.updatePostulation(Commands.POSTULATION_INSERT,postulation_date,cvform_id
,lettreform_id,offre_emploi_id);
       } catch (SQLException sqle) {
              if (cat.isDebugEnabled()) {
                     cat.debug("Error during insert postulation. " +
sqle.getMessage());
              }
              return mapping.findForward("error");
       }

       if (result == 0) {
              if (cat.isDebugEnabled()) {
                     cat.debug("Error insert postulation.");
              }
              return mapping.findForward("error");
       }
      }

      // Report any errors we have discovered back to the original form
      if(!errors.empty()) {
       saveErrors(request, errors);
       return (new ActionForward(mapping.getInput()));
      }

      // Forward control to the edit user registration page
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.35: PostulationForm.java

```
package jobplace.postulation;
```

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import jobplace.common.*;
import jobplace.cv.CVForm;
import jobplace.lettremotivation.LettreMotivationForm;
import jobplace.offre.OffreEmploi;

import java.util.HashMap;
import java.util.Map;
 /**
 * FormBean for Postulation.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/11 $
 */
public class PostulationForm extends ActionForm {

   private String action = null;
   private int postulation_id = 0;
   private String postulation_date = null;
   private CVForm cvform = null;
   private LettreMotivationForm lettreform = null;
   private OffreEmploi offre_emploi= null;
   private HashMap mapping = new HashMap();  //mappings for link parameters

   public String getAction() {
       return action;
   }
   public void setAction(String action) {
       this.action = action;
   }

   public int getPostulation_id() {
       return postulation_id;
   }
   public void setPostulation_id(int postulation_id) {
       this.postulation_id = postulation_id;
   }

   public String getPostulation_date() {
       return postulation_date;
   }

   public void setPostulation_date(String postulation_date) {
       this.postulation_date = postulation_date;
   }

   public CVForm getCvform() {
      return cvform;
   }

   public void setCvform(CVForm cvform) {
      this.cvform = cvform;
   }

   public LettreMotivationForm getLettreform() {
      return lettreform;
   }

   public void setLettreform(LettreMotivationForm lettreform) {
      this.lettreform = lettreform;
   }

   public OffreEmploi getOffre_emploi() {
      return offre_emploi;
   }
```

```
   public void setOffre_emploi(OffreEmploi offre_emploi) {
      this.offre_emploi = offre_emploi;
   }

   /**
    * The Mapping HashMap that is passed to the LinkTag in the form
    * tag library.  The HashMap is a collection of parameters that will
    * be used to make a query string and add it to the link.
    */
   public void setMapping() {
      mapping.put(Constants.POSTULATION_ID, new Integer(postulation_id));
   }

   public Map getMapping() {
      return mapping;
   }

   public void reset(ActionMapping mapping, HttpServletRequest request) {
      action = "Step-Postulation";
      offre_emploi = new OffreEmploi();
      cvform = new CVForm();
      lettreform = new LettreMotivationForm();

   }

   public ActionErrors validate(ActionMapping mapping,
                            HttpServletRequest request) {
      ActionErrors errors = new ActionErrors();

      if(cvform == null) {
         errors.add("cvform",
             new ActionError("error.cvform.required"));
      }
      if(lettreform == null) {
         errors.add("lettreform",
             new ActionError("error.lettreform.required"));
      }
      if(offre_emploi == null) {
         errors.add("offre_emploi",
             new ActionError("error.offre_emploi.required"));
      }

      return errors;
   }
}
```

D.36: preparePostulationAction.java

```
package jobplace.postulation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
```

```
import jobplace.common.*;
import jobplace.sql.*;
import jobplace.cv.*;
import jobplace.lettremotivation.LettreMotivationForm;
import jobplace.offre.OffreEmploi;


// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * PostulationAction is invoked to preparing a postulation.<br>
 * It gets a reference on the selected offre_emploi. Checks if the candidat has got
already a CV,
 * if not the user is asked to create one. It propose the existing
lettre_motivation to the candidat
 * who has to choose one. The objects are saved in an instance of the
PostulationForm. Its field action
 * is used to describe the current step.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/11 $
 */
public class preparePostulationAction extends Action {

      static Category cat =
Category.getInstance(preparePostulationAction.class.getName());

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
   throws IOException, ServletException {


      // setup method variables
      HttpSession session = request.getSession();
      String action = null;
      String username = null;
      String forward = "continue";
      int candidat_id = 0;
      int result = 0;

      Candidat candidat = (Candidat) session.getAttribute(Constants.CANDIDAT_KEY);
      OffreEmploi offre = null;
      CVForm cvform = null;

      PostulationForm postulation = (PostulationForm)
session.getAttribute(Constants.POSTULATION_KEY);
      // postulationform doesn't exist, create it
      if (postulation == null) {
       postulation = new PostulationForm();
       postulation.setAction("Step-Candidat");
      }
      action = postulation.getAction();

      if(action == null) {
         action = "Step-Candidat";
      }

      if (action.equals("Step-CreateLettreMotivation")) action="Step-
LettreMotivation";

      if(cat.isDebugEnabled()) {
             cat.debug("PostulationAction:  Processing " + action + " action");
      }

      // check if candidat is logged in and the candidat bean exist
      if (candidat==null) {
```

```
              // get username of logged user
              username = request.getRemoteUser();

              // check if user logged in
              if (username == null) {
                     cat.warn("User isn't logged!");
                     return mapping.findForward("error");
              }
              // Populate the candidat bean
              cat.info("Populate candidat bean.");
              // get candidat_id
              try {
                     // get UserID in users
                     candidat_id =
Statements.execQueryUsername("JOBPLACE",Commands.GET_USER_ID,username);
              } catch (SQLException sqle) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("Error getting user id. " + sqle.getMessage());
                     }
              }
              if (candidat_id < 1) {
                     cat.warn("candidat_id is smaller than one.");
                     return mapping.findForward("error");
              }

              candidat = new Candidat();
              // Get Candidat Properties
              try {
                     candidat = (Candidat)
Statements.execQueryBean(candidat,Commands.GET_CANDIDAT_PROPERTIES,candidat_id);
              } catch (SQLException sqle) {
                     if(cat.isDebugEnabled()) {
                            cat.debug("Error getting Candidat properties. " +
sqle.getMessage());
                     }
              }

              if(cat.isDebugEnabled()) {
                            cat.debug("Populated bean candidat: " +
candidat.toString());
              }

              candidat.setCandidat_id(candidat_id);
              session.setAttribute(Constants.CANDIDAT_KEY,candidat);
              action="Step-Offre";
       } else if (action.equals("Step-Candidat")) {
              action="Step-Offre";
       }
       // now candidat is logged in
      candidat_id = candidat.getCandidat_id();

      if(cat.isDebugEnabled()) {
              cat.debug("PostulationAction:  Processing " + action + " action");
      }

      if (action.equals("Step-Offre")) {
       offre = (OffreEmploi) session.getAttribute(Constants.OFFRE_EMPLOI_KEY);
       if (offre == null) {
              String offre_id = request.getParameter(Constants.OFFRE_EMPLOI_ID);
              if (offre_id == null) {
                     cat.warn("No offre_emploi is selected");
                     return mapping.findForward("error");
              }
              int offre_emploi_id = Integer.parseInt(offre_id);
              if (offre_emploi_id < 1) {
                     cat.warn("No offre_emploi is selected");
                     return mapping.findForward("error");
              }
              // get offre_emploi bean
              else {
```

```
                    offre = new OffreEmploi();
                    offre.setOffre_emploi_id(offre_emploi_id);
              }
       }
       if (cat.isDebugEnabled()) {
              cat.debug("Offre_emploi_id = " + offre.getOffre_emploi_id());
       }
       // set offre_emploi bean, only offre_emploi_id is needed
       postulation.setOffre_emploi(offre);
       action="Step-CreateCV";
      } // END: Step-Offre

      if(cat.isDebugEnabled()) {
              cat.debug("PostulationAction:  Processing " + action + " action");
      }

      if (action.equals("Step-CreateCV")) {
       // get CV, if it doesn't exist create it
       cvform = new CVForm();
       candidat_id = candidat.getCandidat_id();
       //Check if user has got already a CV
       try {
              result =
Statements.execQueryId("JOBPLACE",Commands.CHECK_CV,candidat_id);
       } catch (SQLException sqle) {
              if (cat.isDebugEnabled()) {
                    cat.debug("Error during checking if cv already exists or not. "
+ sqle.getMessage());
              }
              return mapping.findForward("error");
       }
       // If result is > 0, then cv exists already. -> change action to "Step-
LettreMotivation"
       // and populate CVForm
       if (result > 0) {

              // populate cv
              try {
                    cvform = (CVForm)
Statements.execQueryBean(cvform,Commands.GET_CV_PROPERTIES,candidat_id);
                    if (cvform == null) {
                          cat.warn("cvform is null");
                          return mapping.findForward("error");
                    }
              } catch (SQLException sqle) {
                    if(cat.isDebugEnabled()) {
                          cat.debug("Error while populating cv. " +
sqle.getMessage());
                    }
                    return mapping.findForward("error");
              }
              // only cv_id and cv_object are needed
                cvform.setCandidat(candidat);
              action="Step-LettreMotivation";
              if (cat.isDebugEnabled()) {
                    cat.debug("Cv_id = " + cvform.getCv_id());
              }
              postulation.setCvform(cvform);
       }
       else if (result!=0) {
              if(cat.isDebugEnabled()) {
                    cat.warn("DB Jobplace, table cv couldn't be checked.");
              }
              // Forward control to the specified success URI
              return mapping.findForward("error");
       }
       // create CV
       else {
              forward = "createCV";
       }
```

```
      }
      if (action.equals("Step-CVSet")) {
       // cvform is in postulation which in the session (POSTULATION_KEY)
       action = "Step-LettreMotivation";
      }

      if(cat.isDebugEnabled()) {
             cat.debug("PostulationAction:  Processing " + action + " action");
      }

      if (action.equals("Step-LettreMotivation")) {
       // get array of lettre_motivation or create one
       LettreMotivationForm[] lettres = null;
       try {
             lettres =
Statements.queryLettres(Commands.GET_LETTRES_MOTIVATION,candidat_id);
       } catch (SQLException sqle) {
             if (cat.isDebugEnabled()) {
                   cat.debug("Error during populating lettres motivation. " +
sqle.getMessage());
             }
             return mapping.findForward("error");
       }
       // if letttres is null then create new lettre_motivation
       if (lettres == null || lettres.length==0) {
             if (cat.isDebugEnabled()) {
                   cat.debug("No lettres, create one.");
             }
             action = "Step-CreateLettreMotivation";
             forward = "createLettre";
       } else {
             // show only the first ten characters of lettre_motivation_texte
             for (int i=0;i<lettres.length;i++) {
                   String tmp = lettres[i].getLettre_motivation_texte();
                   if (tmp.length() > 19) {

       lettres[i].setLettre_motivation_texte_tmp(tmp.substring(0,19));
                   } else lettres[i].setLettre_motivation_texte_tmp(tmp);
             }
             // if lettres not null then save in session
             session.setAttribute(Constants.LETTRE_MOTIVATION_ARRAY_KEY,lettres);
             postulation.setLettreform(new LettreMotivationForm());
             action = "Step-Postulation";
        }
      }

      if (action.equals("Step-LettreMotivationSet")) {
       LettreMotivationForm tmplettre = postulation.getLettreform();
       // show only the first ten characters of lettre_motivation_texte
       String tmp = tmplettre.getLettre_motivation_texte();
       if (tmp.length() > 19) {
             tmplettre.setLettre_motivation_texte_tmp(tmp.substring(0,19));
       } else tmplettre.setLettre_motivation_texte_tmp(tmp);

       action="Step-Postulation";
       postulation.setLettreform(tmplettre);
      }

      postulation.setAction(action);

      if("request".equals(mapping.getScope())) {
             request.setAttribute(mapping.getAttribute(), postulation);
      }

      // save postulationform in session
      session.setAttribute(Constants.POSTULATION_KEY,postulation);

      // Forward control to the edit user registration page
      cat.info(" Forwarding to '" + forward + "' page");
```

```
      return (mapping.findForward(forward));
   }
}
```

D.37: ShowPostulationsAction.java

```
package jobplace.postulation;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;

// Import framework struts classes.
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

import java.sql.SQLException;
import jobplace.sql.*;
import jobplace.common.*;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

 /**
 * Show postulations.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/18 $
 */
public class ShowPostulationsAction extends Action {

      static Category cat =
Category.getInstance(ShowPostulationsAction.class.getName());

      public ActionForward perform (ActionMapping mapping,
                                    ActionForm form,
                                    HttpServletRequest request,
                                    HttpServletResponse response)
            throws IOException, ServletException {

      // setup method variables
      HttpSession session = request.getSession();
      PostulationForm[] postulations = null;
      int offre_emploi_id =
Integer.parseInt(request.getParameter("offre_emploi_id"));

      if(cat.isDebugEnabled()) {
            cat.debug("showPostulationsAction:  Processing.");
      }
      try {
            postulations =
Statements.queryPostulations(Commands.GET_POSTULATIONS_OFFRE,offre_emploi_id);
      } catch (SQLException sqle) {
            if (cat.isDebugEnabled()) {
                  cat.debug("Error during populating postulations. " +
sqle.getMessage());
            }
            return mapping.findForward("error");
      }
```

```
      if (postulations == null || postulations.length==0) {
            postulations = new PostulationForm[0];
            if (cat.isDebugEnabled()) {
                  cat.debug("No postulations.");
            }
      }

      session.setAttribute(Constants.POSTULATIONS_ARRAY_KEY, postulations);

      // Forward control
      cat.info(" Forwarding to 'continue' page");

      return (mapping.findForward("continue"));
   }
}
```

D.38: Commands.java

```
package jobplace.sql;

/**
 * SQL command constants and custom statements
 * for the jobplace package.
 * @author Yvette Feldmann
 * @version $Revision: 1.0 $ $Date: 2002/03/01 $
 */
public final class Commands {

    // ---- SQL Statements ----
    /**
     * Name for USERS table
     */
    public static final String USERS_TABLE = "users ";
    /**
     * Name for USER_ROLES table
     */
    public static final String USER_ROLES_TABLE = "user_roles ";
    /**
     * Name for USERS table
     */
    public static final String USER_JOBPLACE_TABLE = "users ";
    /**
     * Name for CANDIDAT table
     */
    public static final String CANDIDAT_TABLE = "candidat ";
    /**
     * Name for RECRUTEUR table
     */
    public static final String RECRUTEUR_TABLE = "recruteur ";
    /**
     * Name for CV table
     */
    public static final String CV_TABLE = "cv ";
    /**
     * Name for FORMATION table
     */
    public static final String FORMATION_TABLE = "formation ";
    /**
     * Name for CONNAISSANCES_LINGUISTIQUES table
     */
    public static final String CONNAISSANCES_LINGUISTIQUES_TABLE =
"connaissances_linguistiques ";
    /**
     * Name for CONNAISSANCES_INFORMATIQUES table
     */
    public static final String CONNAISSANCES_INFORMATIQUES_TABLE =
"connaissances_informatiques ";
    /**
     * Name for EXPERIENCE_PROFESSIONNELLE table
```

```
     */
    public static final String EXPERIENCE_PROFESSIONNELLE_TABLE =
"experience_professionnelle ";
    /**
     * Name for OFFRE_EMPLOI table
     */
    public static final String OFFRE_EMPLOI_TABLE = "offre_emploi ";
    /**
     * Name for LETTRE_MOTIVATION table
     */
    public static final String LETTRE_MOTIVATION_TABLE = "lettre_motivation ";
    /**
     * Name for POSTULATION table
     */
    public static final String POSTULATION_TABLE = "postulation ";

    // ---- insert ----
    /**
     * Command to insert user into USERS table
     */
    public static final String USER_INSERT = "INSERT INTO "
        + USERS_TABLE +
        "(user_name,user_pass) " +
        "VALUES (?,?)";
    /**
     * Command to insert user into USER_ROLES table
     */
    public static final String USER_ROLES_INSERT = "INSERT INTO "
        + USER_ROLES_TABLE +
        "(user_name,role_name) " +
        "VALUES (?,?)";
    /**
     * Command to insert user into USER_ROLES table
     */
    public static final String USER_JOBPLACE_INSERT = "INSERT INTO "
        + USER_JOBPLACE_TABLE +
        "(user_name,user_pass,user_role) " +
        "VALUES (?,?,?)";
     /**
     * Command to insert candidat into CANDIDAT table
     */
    public static final String CANDIDAT_INSERT = "INSERT INTO "
        + CANDIDAT_TABLE +
"(candidat_titre,candidat_prenom,candidat_nom,candidat_email,candidat_adresse,candi
dat_code_postal,candidat_ville,candidat_pays,candidat_date_naissance,candidat_phone
,candidat_portable,fk_user_id) " +
        "VALUES (?,?,?,?,?,?,?,?,?,?,?,?)";
     /**
     * Command to insert recruteur into RECRUTEUR table
     */
    public static final String RECRUTEUR_INSERT = "INSERT INTO "
        + RECRUTEUR_TABLE +
"(societe_nom,societe_secteur_activite,societe_phone,societe_url,societe_descriptio
n,contact_titre,contact_nom,contact_prenom,contact_email,fk_user_id) " +
        "VALUES (?,?,?,?,?,?,?,?,?,?)";
    /**
     * Command to insert CV properties into CV table
     */
    public static final String CV_INSERT = "INSERT INTO "
        + CV_TABLE +
        "(cv_date,cv_objectif,fk_candidat_id) " +
        "VALUES (?,?,?)";
    /**
     * Command to insert Offre properties into offre_emploi table
     */
    public static final String OFFRE_INSERT = "INSERT INTO "
        + OFFRE_EMPLOI_TABLE +
        "(offre_emploi_date, offre_emploi_poste, offre_emploi_description_poste,
offre_emploi_qualites_requises, offre_emploi_connaissances_techniques,
      offre_emploi_region, fk_recruteur_id) " +
```

```
        "VALUES (?,?,?,?,?,?,?)";
    /**
     * Command to insert formation properties into formation table
     */
    public static final String FORMATION_INSERT = "INSERT INTO "
        + FORMATION_TABLE +
        "(etablissement_nom,formation_duree,diplome_nom,diplome_lieu,fk_cv_id) " +
        "VALUES (?,?,?,?,?)";
    /**
     * Command to insert connaissances_linguistiques properties into formation
      table*/
    public static final String CONNAISSANCES_LINGUISTIQUES_INSERT = "INSERT INTO "
        + CONNAISSANCES_LINGUISTIQUES_TABLE +
        "(langue_nom,langue_niveau,fk_cv_id) " +
        "VALUES (?,?,?)";
    /**
     * Command to insert connaissances_informatiques properties into formation
      table*/
    public static final String CONNAISSANCES_INFORMATIQUES_INSERT = "INSERT INTO "
        + CONNAISSANCES_INFORMATIQUES_TABLE +
        "(logiciels,langages,systemes_exploitations,programmation,fk_cv_id) " +
        "VALUES (?,?,?,?,?)";
    /**
     * Command to insert experience_professionnelle properties into formation table
     */
    public static final String EXPERIENCE_PROFESSIONNELLE_INSERT = "INSERT INTO "
        + EXPERIENCE_PROFESSIONNELLE_TABLE +
"(entreprise_nom,secteur_activite,poste,region,debut_travail,fin_travail,competence
s,fk_cv_id) " +
        "VALUES (?,?,?,?,?,?,?,?)";
     /**
     * Command to insert LettreMotivation properties into lettre_motivation table
     */
    public static final String LETTRE_MOTIVATION_INSERT = "INSERT INTO "
        + LETTRE_MOTIVATION_TABLE +
        "(lettre_motivation_texte,fk_candidat_id) " +
        "VALUES (?,?)";
     /**
     * Command to insert Postulation properties into postulation table
     */
    public static final String POSTULATION_INSERT = "INSERT INTO "
        + POSTULATION_TABLE +
        "(postulation_date,fk_cv_id,fk_lettre_motivation_id,fk_offre_emploi_id) " +
        "VALUES (?,?,?,?)";

    // ---- update ----
    /**
     * Command to update properties in candidat table by foreign key fk_user_id
     */
    public static final String CANDIDAT_UPDATE = "UPDATE "
        + CANDIDAT_TABLE + "SET " +
"candidat_titre=?,candidat_prenom=?,candidat_nom=?,candidat_email=?,candidat_adress
e=?,candidat_code_postal=?,candidat_ville=?,candidat_pays=?,candidat_date_naissance
=?,candidat_phone=?,candidat_portable=? " +
        "WHERE fk_user_id=?;";
    /**
     * Command to update properties in recruteur table by foreign key fk_user_id
     */
    public static final String RECRUTEUR_UPDATE = "UPDATE "
        + RECRUTEUR_TABLE + "SET " +
"societe_nom=?,societe_secteur_activite=?,societe_phone=?,societe_url=?,societe_des
cription=?,contact_titre=?,contact_nom=?,contact_prenom=?,contact_email=?,fk_user_i
d=?;";
    /**
     * Command to update properties in CV table by foreign key fk_candidat_id
     */
    public static final String CV_UPDATE = "UPDATE "
        + CV_TABLE + "SET " +
        "cv_date=?,cv_objectif=? " +
        "WHERE fk_candidat_id=?";
```

```
    /**
     * Command to update Offre properties into offre_emploi table by primary key
offre_emploi_id
     */
    public static final String OFFRE_UPDATE = "UPDATE "
        + OFFRE_EMPLOI_TABLE + "SET " +
        "offre_emploi_date=?, offre_emploi_poste=?,
offre_emploi_description_poste=?, offre_emploi_qualites_requises=?,
offre_emploi_connaissances_techniques=?,offre_emploi_region=? " +
        "WHERE offre_emploi_id=?";
    /**
     * Command to update formation properties into formation table
     */
    public static final String FORMATION_UPDATE = "UPDATE "
        + FORMATION_TABLE + "SET " +
        "etablissement_nom=?,formation_duree=?,diplome_nom=?,diplome_lieu=? " +
        "WHERE formation_id=?";
   /**
            * Command to update connaissances_linguistiques properties into
      formation     */
    public static final String CONNAISSANCES_LINGUISTIQUES_UPDATE = "UPDATE "
        + CONNAISSANCES_LINGUISTIQUES_TABLE + "SET " +
        "langue_nom=?,langue_niveau=? " +
        "WHERE langue_id=?";
    /**
     * Command to update connaissances_informatiques properties into formation
     table*/
    public static final String CONNAISSANCES_INFORMATIQUES_UPDATE = "UPDATE "
        + CONNAISSANCES_INFORMATIQUES_TABLE + "SET " +
        "logiciels=?,langages=?,systemes_exploitations=?,programmation=? " +
        "WHERE informatique_id=?";
    /**
     * Command to update experience_professionnelle properties into formation table
     */
    public static final String EXPERIENCE_PROFESSIONNELLE_UPDATE = "UPDATE "
        + EXPERIENCE_PROFESSIONNELLE_TABLE + "SET " +
"entreprise_nom=?,secteur_activite=?,poste=?,region=?,debut_travail=?,fin_travail=?
,competences=? " +
        "WHERE experience_id=?";
    /**
     * Command to update lettremotivation properties into lettre_motivation table
     */
    public static final String LETTRE_MOTIVATION_UPDATE = "UPDATE "
        + LETTRE_MOTIVATION_TABLE + "SET " +
        "lettre_motivation_texte=? " +
        "WHERE lettre_motivation_id=?";

    // ---- delete / undelete ----
    public static final String OFFRE_EMPLOI_DELETE = "DELETE FROM " +
      OFFRE_EMPLOI_TABLE + " WHERE offre_emploi_id=?";
    public static final String LETTRE_MOTIVATION_DELETE = "DELETE FROM " +
      LETTRE_MOTIVATION_TABLE + " WHERE lettre_motivation_id=?";

    // ---- select ----
    /**
     * Command to get user_id.
     */
    public static final String GET_USER_ID = "SELECT user_id FROM " + USERS_TABLE +
        "WHERE user_name=?;";
    /**
     * Command to get cv_id.
     */
    public static final String GET_CV_ID = "SELECT cv_id FROM " + CV_TABLE +
        "WHERE fk_candidat_id=?;";
    /**
     * Command to check if user_name already exists.
     */
    public static final String CHECK_USERNAME = "SELECT COUNT(user_name) FROM " +
USERS_TABLE +
        "WHERE user_name=?;";
```

```
    /**
     * Command to check if cv already exists.
     */
    public static final String CHECK_CV = "SELECT COUNT(cv_id) FROM " + CV_TABLE +
        "WHERE fk_candidat_id=?;";
    /**
     * Base command to select default properties from USERS table
     */
    public static final String USERS_SELECT_BASE = "SELECT " +
        "user_name" +
        "FROM " + USERS_TABLE;

    // ---- search ----
    /**
     * Base command to select search list from Candidat table
     */
    public static final String CANDIDAT_SEARCH_BASE = "SELECT " +
"candidat_titre,candidat_prenom,candidat_nom,candidat_email,candidat_adresse,candid
at_code_postal,candidat_ville,candidat_pays,candidat_date_naissance,candidat_phone,
candidat_portable " +
        "FROM " + CANDIDAT_TABLE;
    /**
     * Command to select candidat by candidat_id
     */
    public static final String GET_CANDIDAT_PROPERTIES = CANDIDAT_SEARCH_BASE +
        " WHERE fk_user_id=?";
    /**
     * Base command to select search list from recruteur table
     */
    public static final String RECRUTEUR_SEARCH_BASE = "SELECT " +
"societe_nom,societe_secteur_activite,societe_phone,societe_url,societe_description
,contact_titre,contact_nom,contact_prenom,contact_email " +
        "FROM " + RECRUTEUR_TABLE;
    /**
     * Command to select recruteur by recruteur_id
     */
    public static final String GET_RECRUTEUR_PROPERTIES = RECRUTEUR_SEARCH_BASE +
        " WHERE fk_user_id=?";
    /**
     * Base command to select search list from CV table
     */
    public static final String CV_SEARCH_BASE = "SELECT " +
        "cv_id,cv_objectif,cv_date " +
        "FROM " + CV_TABLE;
    /**
     * Command to select cv by fk_candidat_id
     */
    public static final String GET_CV_PROPERTIES = CV_SEARCH_BASE +
        " WHERE fk_candidat_id=?";
    /**
     * Base command to select search list from FORMATION table
     */
    public static final String FORMATION_SEARCH_BASE = "SELECT " +
        "formation_id,etablissement_nom,formation_duree,diplome_nom,diplome_lieu "
      + "FROM " + FORMATION_TABLE;
    /**
     * Command to select formation by fk_cv_id
     */
    public static final String GET_FORMATION_PROPERTIES = FORMATION_SEARCH_BASE +
        " WHERE fk_cv_id=?";
    /**
     * Base command to select search list from CONNAISSANCES_LINGUISTIQUES table
     */
    public static final String CONNAISSANCES_LINGUISTIQUES_SEARCH_BASE = "SELECT "
      + "langue_id,langue_nom,langue_niveau " +
        "FROM " + CONNAISSANCES_LINGUISTIQUES_TABLE;
    /**
     * Command to select formation by fk_cv_id
     */
```

```
    public static final String GET_CONNAISSANCES_LINGUISTIQUES_PROPERTIES =
CONNAISSANCES_LINGUISTIQUES_SEARCH_BASE +
        " WHERE fk_cv_id=?";
    /**
     * Base command to select search list from CONNAISSANCES_INFORMATIQUES table
     */
    public static final String CONNAISSANCES_INFORMATIQUES_SEARCH_BASE = "SELECT "
      + "informatique_id,logiciels,langages,systemes_exploitations,programmation "
      + "FROM " + CONNAISSANCES_INFORMATIQUES_TABLE;
    /**
     * Command to select formation by fk_cv_id
     */
    public static final String GET_CONNAISSANCES_INFORMATIQUES_PROPERTIES =
CONNAISSANCES_INFORMATIQUES_SEARCH_BASE +
        " WHERE fk_cv_id=?";
    /**
     * Base command to select search list from EXPERIENCE_PROFESSIONNELLE table
     */
    public static final String EXPERIENCE_PROFESSIONNELLE_SEARCH_BASE = "SELECT " +
"experience_id,entreprise_nom,secteur_activite,poste,region,debut_travail,fin_trava
il,competences " +
        "FROM " + EXPERIENCE_PROFESSIONNELLE_TABLE;
    /**
     * Command to select formation by fk_cv_id
     */
    public static final String GET_EXPERIENCE_PROFESSIONNELLE_PROPERTIES =
EXPERIENCE_PROFESSIONNELLE_SEARCH_BASE +
        " WHERE fk_cv_id=?";
    /**
     * Base command to select search list from OFFRE_EMPLOI table
     */
    public static final String OFFRE_EMPLOI_SEARCH_BASE = "SELECT " +
"offre_emploi_id,offre_emploi_date,offre_emploi_poste,offre_emploi_description_post
e,offre_emploi_qualites_requises,offre_emploi_connaissances_techniques,offre_emploi
_region,fk_recruteur_id " +
        "FROM " + OFFRE_EMPLOI_TABLE;
    /**
     * Command to select all offres
     */
    public static final String GET_OFFRES_EMPLOI = OFFRE_EMPLOI_SEARCH_BASE;
    /**
     * Command to select all offres for recruteur
     */
    public static final String GET_OFFRES_EMPLOI_RECRUTEUR =
OFFRE_EMPLOI_SEARCH_BASE +
        " WHERE fk_recruteur_id=?";
    /**
     * Base command to select search list from LETTRE_MOTIVATION table
     */
    public static final String LETTRE_MOTIVATION_SEARCH_BASE = "SELECT " +
        "lettre_motivation_id,lettre_motivation_texte,fk_candidat_id " +
        "FROM " + LETTRE_MOTIVATION_TABLE;
    /**
     * Command to select all lettres for candidat_id
     */
    public static final String GET_LETTRES_MOTIVATION =
LETTRE_MOTIVATION_SEARCH_BASE +
       " WHERE fk_candidat_id=?";

    /**
     * Command to select all CVs
     */
    public static final String GET_CVS = "SELECT * FROM ((((candidat INNER JOIN cv
ON candidat.fk_user_id = cv.fk_candidat_id) INNER JOIN formation ON cv.cv_id =
formation.fk_cv_id) INNER JOIN connaissances_linguistiques ON
connaissances_linguistiques.fk_cv_id = cv.cv_id) INNER JOIN
connaissances_informatiques ON connaissances_informatiques.fk_cv_id = cv.cv_id)
INNER JOIN experience_professionnelle ON experience_professionnelle.fk_cv_id =
cv.cv_id;";
    /**
```

```
     * Command to select all postulation for offre_emploi_id
     */
    public static final String GET_POSTULATIONS_OFFRE = "SELECT * FROM
(((((((postulation inner join offre_emploi on
postulation.fk_offre_emploi_id=offre_emploi.offre_emploi_id) " +
       "inner join lettre_motivation on
lettre_motivation.lettre_motivation_id=postulation.fk_lettre_motivation_id) inner
join cv ON postulation.fk_cv_id) inner join candidat on candidat.fk_user_id =
cv.fk_candidat_id) INNER JOIN formation ON cv.cv_id = formation.fk_cv_id) " +
       "INNER JOIN connaissances_linguistiques ON
connaissances_linguistiques.fk_cv_id = cv.cv_id) INNER JOIN
connaissances_informatiques ON connaissances_informatiques.fk_cv_id = cv.cv_id)
INNER JOIN experience_professionnelle ON experience_professionnelle.fk_cv_id =
cv.cv_id " +
       "WHERE offre_emploi.offre_emploi_id=?";
// ---- End Data ----
}
```

D.39: ResultSetUtils.java

```
package jobplace.sql;

import java.lang.reflect.InvocationTargetException;

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Types;

import java.util.ArrayList;
import java.util.HashMap;

import jobplace.common.*;
import jobplace.cv.*;
import jobplace.lettremotivation.LettreMotivationForm;
import jobplace.offre.OffreEmploi;
import jobplace.postulation.PostulationForm;

import org.apache.struts.util.BeanUtils;
// import org.apache.commons.beanutil.BeanUtils;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

/**
 * General purpose utility methods related to ResultSets
 *
 * @author
 * @version $Revision: 1.1 $ $Date:  $
 */
 public class ResultSetUtils {

   static Category cat = Category.getInstance(ResultSetUtils.class.getName());

   /**
     * Populate the properties of the specified JavaBean from the next record
     * of the specified ResultSet, based on matching each column name against the
     * corresponding JavaBeans "property setter" methods in the bean's class.
     * // :TODO:
     * Suitable conversion is done for argument types as described under
     * <code>convert()</code>.
     *
     * @param bean      The JavaBean whose properties are to be set
     * @param resultSet The ResultSet whose parameters are to be used
     *                  to populate bean properties
     *
     * @exception SQLException if an exception is thrown while setting
     *            property values or access the ResultSet
```

```
     */
    public static void populate(Object bean,
                                ResultSet resultSet)
        throws SQLException {

        // Build a list of relevant column properties from this resultSet
        HashMap properties = new HashMap();

        // Acquire resultSet MetaData
        ResultSetMetaData metaData = resultSet.getMetaData();
        int cols = metaData.getColumnCount();

        // Scroll to next record and pump into hashmap
        if (resultSet.next()) for (int i=1; i<=cols ; i++) {
            /*
                :TODO: Let native types through
            int type = metaData.getType(i);
            if ...
            properties.put(metaData.getColumnName(i),
                resultSet.getObject(i));
            else
            */
            properties.put(metaData.getColumnName(i),
                resultSet.getString(i));
            if (cat.isDebugEnabled()) {
             cat.debug("Populate bean. ColumnNo: " + i + ", Name: " +
metaData.getColumnName(i) + ", String: " + resultSet.getString(i));
            }

        } else return; // resultSet has no more line

        // Set the corresponding properties of our bean
        try {
            BeanUtils.populate(bean, properties);
        } catch (Exception e) {
            throw new SQLException("BeanUtils.populate threw " + e.toString());
        }
    }

    /** Populate array of beans
    *   @param type_bean describe the type of the bean to populate (OffreEmploi or
LettreMotivation)
    *  @param resultSet
    */
    public static ArrayList populateArray(String type_bean,
                                ResultSet resultSet)
        throws SQLException {

        // Build a list of relevant column properties from this resultSet
        HashMap properties = new HashMap();

        // Acquire resultSet MetaData
        ResultSetMetaData metaData = resultSet.getMetaData();
        int cols = metaData.getColumnCount();
        Object bean=null;

        // needed to save beans
        ArrayList arraylist = new ArrayList();


        // Scroll to next record and pump into hashmap
        while (resultSet.next()) {

             // detect type of bean and create an instance
              if (type_bean.equals("offre_emploi")) {
                   bean = new OffreEmploi();
              } else if (type_bean.equals("lettre_motivation")) {
                   bean = new LettreMotivationForm();
              }
```

```
             for (int i=1; i<=cols ; i++) {
            /*
                :TODO: Let native types through
            int type = metaData.getType(i);
            if ...
            properties.put(metaData.getColumnName(i),
                resultSet.getObject(i));
            else
            */
                  properties.put(metaData.getColumnName(i),
                      resultSet.getString(i));
                  if (cat.isDebugEnabled()) {
                   cat.debug("Populate bean. ColumnNo: " + i + ", Name: " +
metaData.getColumnName(i) + ", String: " + resultSet.getString(i));
                  }
              }
              // Set the corresponding properties of our bean
              try {
                  BeanUtils.populate(bean, properties);
              } catch (Exception e) {
                  throw new SQLException("BeanUtils.populate threw " +
e.toString());
              }
              if (bean != null) {
                   arraylist.add(bean);
                   if (cat.isDebugEnabled()) {
                         cat.debug("Add bean to array list");
                   }
             }
        }
        return arraylist;
    }

    /** Populate array of cvform
    */
    public static ArrayList populateArrayCVForm(ResultSet resultSet)
        throws SQLException {

        // Build a list of relevant column properties from this resultSet
        HashMap properties = new HashMap();

        // Acquire resultSet MetaData
        ResultSetMetaData metaData = resultSet.getMetaData();
        int cols = metaData.getColumnCount();
        Object bean=null;

        ArrayList arraylist = new ArrayList();

        CVForm cvform = null;
        Formation formation = null;
        ConnaissancesLinguistiques langues = null;
        ConnaissancesInformatiques informatiques = null;
        ExperienceProfessionnelle experience = null;


        // Scroll to next record and pump into hashmap
        while (resultSet.next()) {

               cvform = new CVForm();
             formation = new Formation();
             langues = new ConnaissancesLinguistiques();
             informatiques = new ConnaissancesInformatiques();
             experience = new ExperienceProfessionnelle();

             for (int i=1; i<=cols ; i++) {

            /*
                :TODO: Let native types through
            int type = metaData.getType(i);
            if ...
```

```
            properties.put(metaData.getColumnName(i),
                resultSet.getObject(i));
            else
            */
                  properties.put(metaData.getColumnName(i),
                      resultSet.getString(i));
                  if (cat.isDebugEnabled()) {
                   cat.debug("Populate bean. ColumnNo: " + i + ", Name: " +
metaData.getColumnName(i) + ", String: " + resultSet.getString(i));
                  }
               }
               // Set the corresponding properties of the bean
               try {
                   BeanUtils.populate(cvform, properties);
               } catch (Exception e) {
                   throw new SQLException("BeanUtils.populate cvform threw " +
e.toString());
               }
               // Set the corresponding properties of the bean
               try {
                   BeanUtils.populate(formation, properties);
               } catch (Exception e) {
                   throw new SQLException("BeanUtils.populate formation threw " +
e.toString());
               }
                           // Set the corresponding properties of the bean
               try {
                   BeanUtils.populate(langues, properties);
               } catch (Exception e) {
                   throw new SQLException("BeanUtils.populate langues threw " +
e.toString());
               }
                     // Set the corresponding properties of the bean
               try {
                   BeanUtils.populate(informatiques, properties);
               } catch (Exception e) {
                   throw new SQLException("BeanUtils.populate informatiques threw "
+ e.toString());
               }
                     // Set the corresponding properties of the bean
               try {
                   BeanUtils.populate(experience, properties);
               } catch (Exception e) {
                   throw new SQLException("BeanUtils.populate experience threw " +
e.toString());
               }
               if (cvform != null) {
                    cvform.setFormation(formation);
                    cvform.setProfession(experience);
                    cvform.setLangues(langues);
                    cvform.setInformatiques(informatiques);
                    arraylist.add(cvform);
                    if (cat.isDebugEnabled()) {
                           cat.debug("Add cvform to array list");
                    }
             }
        }
        return arraylist;
    }
    /** Populate array of postulationform
    */
    public static ArrayList populateArrayPostulationForm(ResultSet resultSet)
        throws SQLException {

        // Build a list of relevant column properties from this resultSet
        HashMap properties = new HashMap();

        // Acquire resultSet MetaData
        ResultSetMetaData metaData = resultSet.getMetaData();
        int cols = metaData.getColumnCount();
```

```
        Object bean=null;

        ArrayList arraylist = new ArrayList();

        PostulationForm postulation = null;
        OffreEmploi offre_emploi = null;
        LettreMotivationForm lettre_motivation = null;
        CVForm cvform = null;
        Candidat candidat = null;
        Formation formation = null;
        ConnaissancesLinguistiques langues = null;
        ConnaissancesInformatiques informatiques = null;
        ExperienceProfessionnelle experience = null;


        // Scroll to next record and pump into hashmap
        while (resultSet.next()) {

               postulation = new PostulationForm();
               offre_emploi = new OffreEmploi();
               lettre_motivation = new LettreMotivationForm();
               cvform = new CVForm();
               candidat = new Candidat();
              formation = new Formation();
              langues = new ConnaissancesLinguistiques();
              informatiques = new ConnaissancesInformatiques();
              experience = new ExperienceProfessionnelle();

              for (int i=1; i<=cols ; i++) {

             /*
                 :TODO: Let native types through
             int type = metaData.getType(i);
             if ...
             properties.put(metaData.getColumnName(i),
                 resultSet.getObject(i));
             else
             */
                    properties.put(metaData.getColumnName(i),
                        resultSet.getString(i));
                    if (cat.isDebugEnabled()) {
                     cat.debug("Populate bean. ColumnNo: " + i + ", Name: " +
metaData.getColumnName(i) + ", String: " + resultSet.getString(i));
                    }
                }

                // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(postulation, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate cvform threw " +
e.toString());
                }
                // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(offre_emploi, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate cvform threw " +
e.toString());
                }
                // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(lettre_motivation, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate cvform threw " +
e.toString());
                }
                // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(cvform, properties);
```

```
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate cvform threw " +
e.toString());
                }
                // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(candidat, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate cvform threw " +
e.toString());
                }
                // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(formation, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate formation threw " +
e.toString());
                }
                            // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(langues, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate langues threw " +
e.toString());
                }
                   // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(informatiques, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate informatiques threw "
+ e.toString());
                }
                   // Set the corresponding properties of the bean
                try {
                    BeanUtils.populate(experience, properties);
                } catch (Exception e) {
                    throw new SQLException("BeanUtils.populate experience threw " +
e.toString());
                }
                if (cvform != null) {
                     cvform.setCandidat(candidat);
                     cvform.setFormation(formation);
                     cvform.setProfession(experience);
                     cvform.setLangues(langues);
                     cvform.setInformatiques(informatiques);
               }
               if (postulation != null) {
                     postulation.setCvform(cvform);
                     postulation.setLettreform(lettre_motivation);
                     postulation.setOffre_emploi(offre_emploi);
                     arraylist.add(postulation);
                     if (cat.isDebugEnabled()) {
                           cat.debug("Add postulation to array list");
                     }
               }
          }
          return arraylist;
     }
}
```

D.40: Statements.java

```
package jobplace.sql;

import java.util.ArrayList;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;

import org.apache.scaffold.sql.ConnectionPool;

import jobplace.lettremotivation.LettreMotivationForm;
import jobplace.common.*;
import jobplace.cv.*;
import jobplace.postulation.PostulationForm;
import jobplace.offre.OffreEmploi;

// Import log4j classes for logging and debugging.
import org.apache.log4j.Category;
import org.apache.log4j.PropertyConfigurator;

/**
 * SQL statements for the jobplace package.
 * @author Yvette Feldmann-Ossele Atangana
 * @version $Revision: 1.0 $ $Date: 2002/03/01 $
 */
public final class Statements {

       static Category cat = Category.getInstance(Statements.class.getName());

    // ---- Article SQL Statements ----

        /**
         * Execute given command.
         * Query on user_name.
         * <p>
         * @return -1 if fails without an exception
         * @exception SQLException if SQL error occurs
         * @param db Database name
         * @param command The SQL statement to execute
         * @param user_name

         * @exception SQLException on SQL error.
        **/
        public static final int execQueryUsername (String db, String command,
                 String user_name
            ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            ResultSet resultSet = null;

            int result = -1;
            try {
                connection = ConnectionPool.getConnection(db);
                statement = connection.prepareStatement(command);
                statement.setObject(1, user_name);

                resultSet = statement.executeQuery();

                if (resultSet.next()) {
                    result = resultSet.getInt(1);
                }
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }
            return result;
        }

        /**
         * Execute given command.
```

```
 * Query on id.
 * <p>
 * @return -1 if fails without an exception
 * @exception SQLException if SQL error occurs
 * @param db Database name
 * @param command The SQL statement to execute
 * @param id

 * @exception SQLException on SQL error.
**/
public static final int execQueryId (String db, String command,
         int id
    ) throws SQLException {

    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    int result = -1;
    try {
        connection = ConnectionPool.getConnection(db);
        statement = connection.prepareStatement(command);
        statement.setInt(1, id);

        resultSet = statement.executeQuery();

        if (resultSet.next()) {
            result = resultSet.getInt(1);
        }
    } // end try

    finally {
        try {
            if (statement != null) statement.close();
            if (connection!= null) connection.close();
        }
        catch (SQLException sqle) {;}
    }
    return result;
}

/**
 * Execute given command.
 * Populate bean with data corresponding to id key
 * <p>
 * @return Object bean
 * @exception SQLException if SQL error occurs
 * @param Object - bean to populate
 * @param command The SQL statement to execute
 * @param id The key
 *
 * @exception SQLException on SQL error.
**/
public static final Object execQueryBean (Object bean, String command,
         int id
    ) throws SQLException {

    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        connection = ConnectionPool.getConnection("JOBPLACE");
        statement = connection.prepareStatement(command);
        statement.setInt(1, id);

        resultSet = statement.executeQuery();

        //populate bean
        try {
            ResultSetUtils.populate(bean,resultSet);
        } catch (SQLException sqle) {
```

```
                    if (cat.isDebugEnabled()) {
                          cat.debug("Error while populating bean. " +
sqle.getMessage());
                    }
                }
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }
            return bean;
        }

        /**
         * Execute given command.
         * Populate array with offres-emploi
         * <p>
         * @return Array of OffreEmploi bean
         * @exception SQLException if SQL error occurs
         * @param command The SQL statement to execute
         * @param id fk_recruteur_id (0 for all offres)
         *
         * @exception SQLException on SQL error.
        **/
        public static final OffreEmploi[] queryOffres (String command, int id
            ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            ResultSet resultSet = null;

            ArrayList offresArrayList = null;
            OffreEmploi offre = null;
            OffreEmploi[] offresArray = new OffreEmploi[0];

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);

                // if id == 0 then get all offres
                if (id != 0) {
                    statement.setInt(1,id);
                }

                resultSet = statement.executeQuery();

                //populate ArrayList
                offre = new OffreEmploi();
                try {
                    offresArrayList =
ResultSetUtils.populateArray("offre_emploi",resultSet);
                } catch (SQLException sqle) {
                    if (cat.isDebugEnabled()) {
                          cat.debug("Error while populating bean. " +
sqle.getMessage());
                    }
                }
                if (offresArrayList != null) {
                    // set mapping for link
                    for (int i=0;i < offresArrayList.size();i++) {
                          offre = (OffreEmploi) offresArrayList.get(i);
                          offre.setMapping();
                          if (cat.isDebugEnabled()) {
                                cat.debug("Ref offre_emploi= " +
offre.getOffre_emploi_id());
                          }
```

```
                    }
                    // copy arraylist in array with correct length
                    offresArray =
(OffreEmploi[])offresArrayList.toArray(offresArray);
                    if (cat.isDebugEnabled()) {
                        cat.debug("Length: offresArrayList= " +
offresArrayList.size() + ", offresArray= " + offresArray.length);
                    }
                }


            } // end try



            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }

            return offresArray;
        }
        /**
         * Execute given command.
         * Populate array with CVs
         * <p>
         * @return Array of CVForm bean
         * @exception SQLException if SQL error occurs
         * @param command The SQL statement to execute
         *
         * @exception SQLException on SQL error.
        **/
        public static final CVForm[] queryCVS (String command
            ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            ResultSet resultSet = null;

            ArrayList cvsArrayList = null;
            CVForm cvform = null;
            CVForm[] cvsArray = new CVForm[0];

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);

                resultSet = statement.executeQuery();

                //populate ArrayList
                cvform = new CVForm();
                try {
                    cvsArrayList = ResultSetUtils.populateArrayCVForm(resultSet);
                } catch (SQLException sqle) {
                    if (cat.isDebugEnabled()) {
                        cat.debug("Error while populating bean. " +
sqle.getMessage());
                    }
                }
                if (cvsArrayList != null) {
                    // set mapping for link
                    for (int i=0;i < cvsArrayList.size();i++) {
                        cvform = (CVForm) cvsArrayList.get(i);
                        cvform.setMapping();
                        if (cat.isDebugEnabled()) {
                            cat.debug("Ref cvform= " + cvform.getCv_id());
                        }
```

```
                    }
                    // copy arraylist in array with correct length
                    cvsArray = (CVForm[])cvsArrayList.toArray(cvsArray);
                    if (cat.isDebugEnabled()) {
                           cat.debug("Length: cvsArrayList= " + cvsArrayList.size()
+ ", cvsArray= " + cvsArray.length);
                    }
                }
            } // end try
            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }

            return cvsArray;
        }
        /**
         * Execute given command.
         * Populate array with postulations
         * <p>
         * @return Array of PostulationForm bean
         * @exception SQLException if SQL error occurs
         * @param command The SQL statement to execute
         * @param offre_emploi_id
         * @exception SQLException on SQL error.
        **/
        public static final PostulationForm[] queryPostulations (String command,
int offre_emploi_id
            ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            ResultSet resultSet = null;

            ArrayList postulationArrayList = null;
            PostulationForm postulation = null;
            PostulationForm[] postulationArray = new PostulationForm[0];

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);

                statement.setInt(1,offre_emploi_id);

                resultSet = statement.executeQuery();

                //populate ArrayList
                postulation = new PostulationForm();
                try {
                    postulationArrayList =
ResultSetUtils.populateArrayPostulationForm(resultSet);
                } catch (SQLException sqle) {
                    if (cat.isDebugEnabled()) {
                           cat.debug("Error while populating bean. " +
sqle.getMessage());
                    }
                }
                if (postulationArrayList != null) {
                    // set mapping for link
                    for (int i=0;i < postulationArrayList.size();i++) {
                           postulation = (PostulationForm)
postulationArrayList.get(i);
                           postulation.setMapping();
                           if (cat.isDebugEnabled()) {
                                  cat.debug("Ref postulationform= " +
postulation.getPostulation_id());
                           }
```

```
                    }
                    // copy arraylist in array with correct length
                    postulationArray =
(PostulationForm[])postulationArrayList.toArray(postulationArray);
                    if (cat.isDebugEnabled()) {
                          cat.debug("Length: postulationArrayList= " +
postulationArrayList.size() + ", postulationArray= " + postulationArray.length);
                    }
                }
            } // end try
            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }

            return postulationArray;
        }
        /**
         * Execute given command.
         * Populate array with lettres-motivation
         * <p>
         * @return Array of LettreMotivationForm bean
         * @exception SQLException if SQL error occurs
         * @param command The SQL statement to execute
         * @param fk_candidat_id
         *
         * @exception SQLException on SQL error.
        **/
        public static final LettreMotivationForm[] queryLettres (String command,
            int fk_candidat_id) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            ResultSet resultSet = null;

            ArrayList lettresArrayList = null;
            LettreMotivationForm lettre = null;
            LettreMotivationForm[] lettresArray = new LettreMotivationForm[0];

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);
                statement.setInt(1,fk_candidat_id);

                if (cat.isDebugEnabled()) {
                    cat.debug(statement.toString());
                }


                resultSet = statement.executeQuery();

                //populate ArrayList
                lettre = new LettreMotivationForm();
                try {
                    lettresArrayList =
ResultSetUtils.populateArray("lettre_motivation",resultSet);
                } catch (SQLException sqle) {
                    if (cat.isDebugEnabled()) {
                          cat.debug("Error while populating bean. " +
sqle.getMessage());
                    }
                }
                if (lettresArrayList != null) {
                    // set mapping for link
                    for (int i=0;i < lettresArrayList.size();i++) {
                          lettre = (LettreMotivationForm) lettresArrayList.get(i);
                          lettre.setMapping();
```

```
                    }
                    // copy arraylist in array with correct length
                    lettresArray =
(LettreMotivationForm[])lettresArrayList.toArray(lettresArray);
                }
            } // end try
            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }

            return lettresArray;
        }

        /**
         * Update users.
         * <p>
         * @return -1 if fails without an exception
         * @exception SQLException if SQL error occurs
         * @param user_name
         * @param param2 (user_pass or user_role)
         *
         * @exception SQLException on SQL error.
        **/
        public static final int updateUsers (String command,
                 String user_name, String param2
            ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            int result = 0;

            try {
                connection = ConnectionPool.getConnection("USERS");
                statement = connection.prepareStatement(command);
                statement.setObject(1, user_name);
                // user_pass or user_role
                statement.setObject(2, param2);

                result = statement.executeUpdate();
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }
            return result;
        }

        /**
         * Update user in JobPlace
         * <p>
         * @return 0 if fails without an exception
         * @exception SQLException if SQL error occurs
         * @param user_name
         * @param user_pass
         * @param user_role
         *
         * @exception SQLException on SQL error.
        **/
        public static final int updateUserJobplace (String command,
                 String user_name, String user_pass, String user_role
            ) throws SQLException {
```

```
    Connection connection = null;
    PreparedStatement statement = null;
    int result = 0;

    try {
        connection = ConnectionPool.getConnection("JOBPLACE");
        statement = connection.prepareStatement(command);
        statement.setObject(1, user_name);
        statement.setObject(2, user_pass);
        statement.setObject(3, user_role);


        result = statement.executeUpdate();
    } // end try

    finally {
        try {
            if (statement != null) statement.close();
            if (connection!= null) connection.close();
        }
        catch (SQLException sqle) {;}
    }
    return result;
}

/**
 * Update candidat.
 * <p>
 * @return 0 if fails without an exception
 * @exception SQLException if SQL error occurs
 * @param candidat bean
 *
 * @exception SQLException on SQL error.
**/
public static final int updateCandidat (String command,
         Candidat candidat
    ) throws SQLException {

    Connection connection = null;
    PreparedStatement statement = null;
    int result = 0;

    try {
        connection = ConnectionPool.getConnection("JOBPLACE");
        statement = connection.prepareStatement(command);
        statement.setObject(1, candidat.getCandidat_titre());
        statement.setObject(2, candidat.getCandidat_prenom());
        statement.setObject(3, candidat.getCandidat_nom());
        statement.setObject(4, candidat.getCandidat_email());
        statement.setObject(5, candidat.getCandidat_adresse());
        statement.setObject(6, candidat.getCandidat_code_postal());
        statement.setObject(7, candidat.getCandidat_ville());
        statement.setObject(8, candidat.getCandidat_pays());
        statement.setObject(9, candidat.getCandidat_date_naissance());
        statement.setObject(10, candidat.getCandidat_phone());
        statement.setObject(11, candidat.getCandidat_portable());
        statement.setInt(12, candidat.getCandidat_id());

       if (cat.isDebugEnabled()) {
            cat.debug(statement.toString());
        }

        result = statement.executeUpdate();
    } // end try

    finally {
        try {
            if (statement != null) statement.close();
            if (connection!= null) connection.close();
        }
```

```
            catch (SQLException sqle) {;}
        }
        return result;
    }

   /**
     * Update recruteur.
     * <p>
     * @return 0 if fails without an exception
     * @exception SQLException if SQL error occurs
     * @param recruteur bean
     *
     * @exception SQLException on SQL error.
    **/
    public static final int updateRecruteur (String command,
             Recruteur recruteur
        ) throws SQLException {

        Connection connection = null;
        PreparedStatement statement = null;
        int result = 0;

        try {
            connection = ConnectionPool.getConnection("JOBPLACE");
            statement = connection.prepareStatement(command);
            statement.setObject(1, recruteur.getSociete_nom());
            statement.setObject(2, recruteur.getSociete_secteur_activite());
            statement.setObject(3, recruteur.getSociete_phone());
            statement.setObject(4, recruteur.getSociete_url());
            statement.setObject(5, recruteur.getSociete_description());
            statement.setObject(6, recruteur.getContact_titre());
            statement.setObject(7, recruteur.getContact_nom());
            statement.setObject(8, recruteur.getContact_prenom());
            statement.setObject(9, recruteur.getContact_email());
            statement.setInt(10, recruteur.getFk_user_id());


           if (cat.isDebugEnabled()) {
                cat.debug(statement.toString());
             }

            result = statement.executeUpdate();
        } // end try

        finally {
            try {
                if (statement != null) statement.close();
                if (connection!= null) connection.close();
            }
            catch (SQLException sqle) {;}
        }
        return result;
    }
   /**
     * Update cv.
     * <p>
     * @return 0 if fails without an exception
     * @exception SQLException if SQL error occurs
     * @param cv_date
     * @param cv_object
     * @param candidat_id
     * @exception SQLException on SQL error.
    **/
    public static final int updateCV (String command,
             String cv_date, String cv_objectif, int fk_candidat_id
        ) throws SQLException {

        Connection connection = null;
        PreparedStatement statement = null;
        int result = 0;
```

```
            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);
                statement.setObject(1, cv_date);
                statement.setObject(2, cv_objectif);
                statement.setInt(3, fk_candidat_id);

               if (cat.isDebugEnabled()) {
                    cat.debug(statement.toString());
                }

                result = statement.executeUpdate();
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }
            return result;
        }
        /**
         * Update offre.
         * <p>
         * @return 0 if fails without an exception
         * @exception SQLException if SQL error occurs
         * @param offre_emploi_date
         * @param offre_emploi_poste
         * @param offre_emploi_description_poste
         * @param offre_emploi_qualites_requises
         * @param offre_emploi_connaissances_techniques
         * @param offre_emploi_region
         * @param id
         * @exception SQLException on SQL error.
        **/
        public static final int updateOffre (String command,
                 String offre_emploi_date,String offre_emploi_poste,String
offre_emploi_description_poste,String offre_emploi_qualites_requises,String
offre_emploi_connaissances_techniques,String offre_emploi_region,int id)
                  throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            int result = 0;

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);
                statement.setObject(1, offre_emploi_date);
                statement.setObject(2, offre_emploi_poste);
                statement.setObject(3, offre_emploi_description_poste);
                statement.setObject(4, offre_emploi_qualites_requises);
                statement.setObject(5, offre_emploi_connaissances_techniques);
                statement.setObject(6, offre_emploi_region);
                statement.setInt(7, id);

               if (cat.isDebugEnabled()) {
                    cat.debug(statement.toString());
                }

                result = statement.executeUpdate();
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
```

```
                }
                catch (SQLException sqle) {;}
            }
            return result;
        }
        /**
         * Update postulation.
         * <p>
         * @return 0 if fails without an exception
         * @exception SQLException if SQL error occurs
         * @param fk_cv_id
         * @param fk_lettre_motivation_id
         * @param fk_offre_emploi_id
         * @exception SQLException on SQL error.
        **/
        public static final int updatePostulation (String command,
                 String postulation_date,int fk_cv_id,int
fk_lettre_motivation_id,int fk_offre_emploi_id
                 ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            int result = 0;

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);
                statement.setObject(1, postulation_date);
                statement.setInt(2, fk_cv_id);
                statement.setInt(3, fk_lettre_motivation_id);
                statement.setInt(4, fk_offre_emploi_id);

               if (cat.isDebugEnabled()) {
                    cat.debug(statement.toString());
                }

                result = statement.executeUpdate();
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }
            return result;
        }
        /**
         * Update lettre_motivation.
         * <p>
         * @return 0 if fails without an exception
         * @exception SQLException if SQL error occurs
         * @param lettre_motivation_texte
         * @param id (fk_candidat_id or lettre_motivation_id)
         * @exception SQLException on SQL error.
        **/
        public static final int updateLettreMotivation (String command,
                 String lettre_motivation_texte, int id
            ) throws SQLException {

            Connection connection = null;
            PreparedStatement statement = null;
            int result = 0;

            try {
                connection = ConnectionPool.getConnection("JOBPLACE");
                statement = connection.prepareStatement(command);
                statement.setObject(1, lettre_motivation_texte);
                statement.setInt(2, id);
```

```
       if (cat.isDebugEnabled()) {
            cat.debug(statement.toString());
         }

         result = statement.executeUpdate();
    } // end try

    finally {
        try {
            if (statement != null) statement.close();
            if (connection!= null) connection.close();
        }
        catch (SQLException sqle) {;}
    }
    return result;
}
/**
 * Update formation.
 * <p>
 * @return 0 if fails without an exception
 * @exception SQLException if SQL error occurs
 * @param formation bean
 * @param id (cv_id or formation_id)
 * @exception SQLException on SQL error.
**/
public static final int updateFormation (String command,
         Formation formation, int id
    ) throws SQLException {

    Connection connection = null;
    PreparedStatement statement = null;
    int result = 0;

    try {
        connection = ConnectionPool.getConnection("JOBPLACE");
        statement = connection.prepareStatement(command);
        statement.setObject(1, formation.getEtablissement_nom());
        statement.setObject(2, formation.getFormation_duree());
        statement.setObject(3, formation.getDiplome_nom());
        statement.setObject(4, formation.getDiplome_lieu());
        statement.setInt(5, id);

       if (cat.isDebugEnabled()) {
            cat.debug(statement.toString());
         }

         result = statement.executeUpdate();
    } // end try

    finally {
        try {
            if (statement != null) statement.close();
            if (connection!= null) connection.close();
        }
        catch (SQLException sqle) {;}
    }
    return result;
}
/**
 * Update connaissances_linguistiques.
 * <p>
 * @return 0 if fails without an exception
 * @exception SQLException if SQL error occurs
 * @param connaissanceslinguistiques bean
 * @param id (cv_id or langue_id)
 * @exception SQLException on SQL error.
**/
public static final int updateLangues (String command,
         ConnaissancesLinguistiques langues, int id
```

```
        ) throws SQLException {

        Connection connection = null;
        PreparedStatement statement = null;
        int result = 0;

        try {
            connection = ConnectionPool.getConnection("JOBPLACE");
            statement = connection.prepareStatement(command);
            statement.setObject(1, langues.getLangue_nom());
            statement.setObject(2, langues.getLangue_niveau());
            statement.setInt(3, id);

           if (cat.isDebugEnabled()) {
                cat.debug(statement.toString());
            }

            result = statement.executeUpdate();
        } // end try

        finally {
            try {
                if (statement != null) statement.close();
                if (connection!= null) connection.close();
            }
            catch (SQLException sqle) {;}
        }
        return result;
    }
    /**
     * Update connaissances_informatiques.
     * <p>
     * @return 0 if fails without an exception
     * @exception SQLException if SQL error occurs
     * @param connaissanceinformatiques bean
     * @param id (cv_id or informatique_id)
     * @exception SQLException on SQL error.
    **/
    public static final int updateInformatiques (String command,
             ConnaissancesInformatiques informatiques, int id
        ) throws SQLException {

        Connection connection = null;
        PreparedStatement statement = null;
        int result = 0;

        try {
            connection = ConnectionPool.getConnection("JOBPLACE");
            statement = connection.prepareStatement(command);
            statement.setObject(1, informatiques.getLogiciels());
            statement.setObject(2, informatiques.getLangages());
            statement.setObject(3, informatiques.getSystemes_exploitations());
            statement.setObject(4, informatiques.getProgrammation());
            statement.setInt(5, id);

           if (cat.isDebugEnabled()) {
                cat.debug(statement.toString());
            }

            result = statement.executeUpdate();
        } // end try

        finally {
            try {
                if (statement != null) statement.close();
                if (connection!= null) connection.close();
            }
            catch (SQLException sqle) {;}
        }
        return result;
```

```
    }
    /**
     * Update experience_professionnelle.
     * <p>
     * @return 0 if fails without an exception
     * @exception SQLException if SQL error occurs
     * @param experienceprofessionnelle bean
     * @param id (cv_id or profession_id)
     * @exception SQLException on SQL error.
    **/
    public static final int updateProfession (String command,
              ExperienceProfessionnelle profession, int id
        ) throws SQLException {

        Connection connection = null;
        PreparedStatement statement = null;
        int result = 0;

        try {
            connection = ConnectionPool.getConnection("JOBPLACE");
            statement = connection.prepareStatement(command);
            statement.setObject(1, profession.getEntreprise_nom());
            statement.setObject(2, profession.getSecteur_activite());
            statement.setObject(3, profession.getPoste());
            statement.setObject(4, profession.getRegion());
            statement.setObject(5, profession.getDebut_travail());
            statement.setObject(6, profession.getFin_travail());
            statement.setObject(7, profession.getCompetences());
            statement.setInt(8, id);

           if (cat.isDebugEnabled()) {
                cat.debug(statement.toString());
            }

            result = statement.executeUpdate();
        } // end try

        finally {
            try {
                if (statement != null) statement.close();
                if (connection!= null) connection.close();
            }
            catch (SQLException sqle) {;}
        }
        return result;
    }
   /**
     * Method for deleting data in table.
     * <p>
     * @return 0 if fails without an exception
     * @exception SQLException if SQL error occurs
     * @param command
     * @param id (which data is to delete)
    **/
    public static final int deleteQuery (String command,
              int id
        ) throws SQLException {

        Connection connection = null;
        PreparedStatement statement = null;
        int result = 0;

        try {
            connection = ConnectionPool.getConnection("JOBPLACE");
            statement = connection.prepareStatement(command);
            statement.setInt(1, id);

           if (cat.isDebugEnabled()) {
                cat.debug(statement.toString());
            }
```

```
                result = statement.executeUpdate();
            } // end try

            finally {
                try {
                    if (statement != null) statement.close();
                    if (connection!= null) connection.close();
                }
                catch (SQLException sqle) {;}
            }
            return result;
        }
// ---- End Statements ----
}
```

Annexe E: CD-ROM

Le CD-ROM contient les répertoires et les fichiers suivants:

- Le répertoire `jobplace-install` qui contient les fichiers utilisés pour l'installation (cf. chapitre 7).
- Le répertoire `jobplace-source` qui contient les sources de l'application.
- La documentation html généré par javadoc qui se trouve dans le répertoire `javadoc`.
- Le rapport au format Word 2000 (`rapport.doc`).
- Le rapport au format PDF (`rapport.pdf`).