Destinataires :
James ASHTON
Max CHEVALIER

Auteure :
Ophélie FRAISIER

10 Avril 2012 - 22 Juin 2012

Rapport de stage

Maintenance, création et amélioration de sites web métiers

Institut d'Études Politiques
2 ter rue des Puits creusés
BP 88526
31685 TOULOUSE cedex 6

Destinataires :
James ASHTON
Max CHEVALIER

Auteure :
Ophélie FRAISIER

10 Avril 2012 - 22 Juin 2012

Rapport de stage

Maintenance, création et amélioration de sites web métiers

Institut d'Études Politiques
2 ter rue des Puits creusés
BP 88526
31685 TOULOUSE cedex 6

Remerciements

Je tiens à remercier ici M. Philippe Raimbault, président de l'IEP Toulouse, de m'avoir accepté dans son établissement.

Merci à James, mon maître de stage, pour ses conseils et son assistance tout au long de ce stage.

Merci à M. Max Chevalier, mon tuteur, pour ses conseils professionnels et vestimentaires.

Merci à Naima et Véronique d'avoir été d'aussi agréables collaboratrices.

Merci à tous mes collègues du troisième étage et plus largement à tous les membres de l'IEP que j'ai eu la chance de croiser régulièrement pour m'avoir permis de travailler dans une ambiance agréable tout au long de ce stage.

Merci enfin à Clément et Leeloo de m'avoir aidé lors de la rédaction de ce rapport.

Table des matières

Introduction

En fin de seconde année de DUT Informatique j'ai du effectuer un stage de 11 semaines en milieu professionnel afin de compléter ma formation. Par l'intermédiaire de Mme Bensadoun, j'ai eu connaissance d'un stage proposé par l'Institut d'Études Politiques de Toulouse qui consistait en la création d'une application web à l'aide du framework php Symfony et en l'administration de bases de données avec mySQL. Souhaitant poursuivre mes études dans le domaine des bases de données et ayant déjà entendu parler de Symfony sans jamais avoir eu l'occasion de l'utiliser j'ai présenté ma candidature et ait été retenue.

Au cours du stage, ma mission initiale a quelque peu été modifiée. Il a fallu que je modifie certains des nombreux outils web dont l'IEP dispose afin d'aider les différents services administratifs dans leurs tâches, pour qu'ils correspondent au mieux aux besoins des utilisateurs. Afin de comprendre pourquoi ces modifications étaient nécessaires je vous présenterai dans le premier chapitre mes conditions de travail durant ce stage, notamment l'organisation de l'IEP et les différents collaborateurs avec qui j'ai été amenée à travailler.

J'ai eu la chance durant ce stage d'utiliser différentes méthodes pour récupérer les données nécessaires à mes applications :

- la consultation d'un annuaire LDAP,
- la gestion d'une base de données relationnelles avec JDBC,
- la création et la gestion d'une base de données relationnelles avec un ORM.

Ces trois aspects seront donc approfondis dans les chapitres suivants, présentant le travail que j'ai effectué tout au long de ce stage, tout d'abord sur PStage, puis sur Spot et le site du Congrès RCSL 2013, en collaboration avec les différents intervenants de chaque projet.

Chapitre 1

Conditions de travail

1.1 Emploi du temps

1.1.1 Horaires hebdomadaires

Occupant pour ce stage un poste à temps complet, je devais réaliser 35 heures de travail hebdomadaire. Mes horaires étaient donc 09h00 - 12h30 et 13h30 - 17h00 du Lundi au Vendredi. Ces horaires pouvaient être modifiés si besoin, avec l'accord du maître de stage.

1.1.2 Gantt du stage

Mon stage a été réparti en trois projets distincts, le déploiement de PStage, l'amélioration de Spot et la création du site d'inscription du Congrès RCSL 2013.

Vous trouverez sur la page suivante le diagramme de Gantt illustrant la répartition temporelle de ces projets *(Figure 1.1)*.

1.2 Contexte professionnel

1.2.1 Présentation de l'entreprise

L'Institut d'Études Politiques de Toulouse (aussi connu sous le nom de Sciences Po Toulouse) est un établissement administratif d'enseignement supérieur membre de la FNSP [1] et dépendant de l'Université Toulouse 1 Capitole. Créé en 1948, il dispense des cours en sciences humaines et sociales à plus de 1500 étudiants.

Il est présidé depuis le 9 Avril 2010 par M. Philippe Raimbault, professeur des universités en Droit public et diplomé de cet IEP en 1995, qui est ainsi devenu le plus jeune directeur de France des IEP.

L'organisation administrative de l'établissement est présentée dans la *figure 1.2*. J'ai pour ma part été principalement en relation durant ce stage avec les pôles «Développement» et «Formations» par l'intermédiaire des services de «Gestion des stages et certification professionnelles», de «Scolarité du Diplôme et du Master» et de «Recherche».

Le service informatique

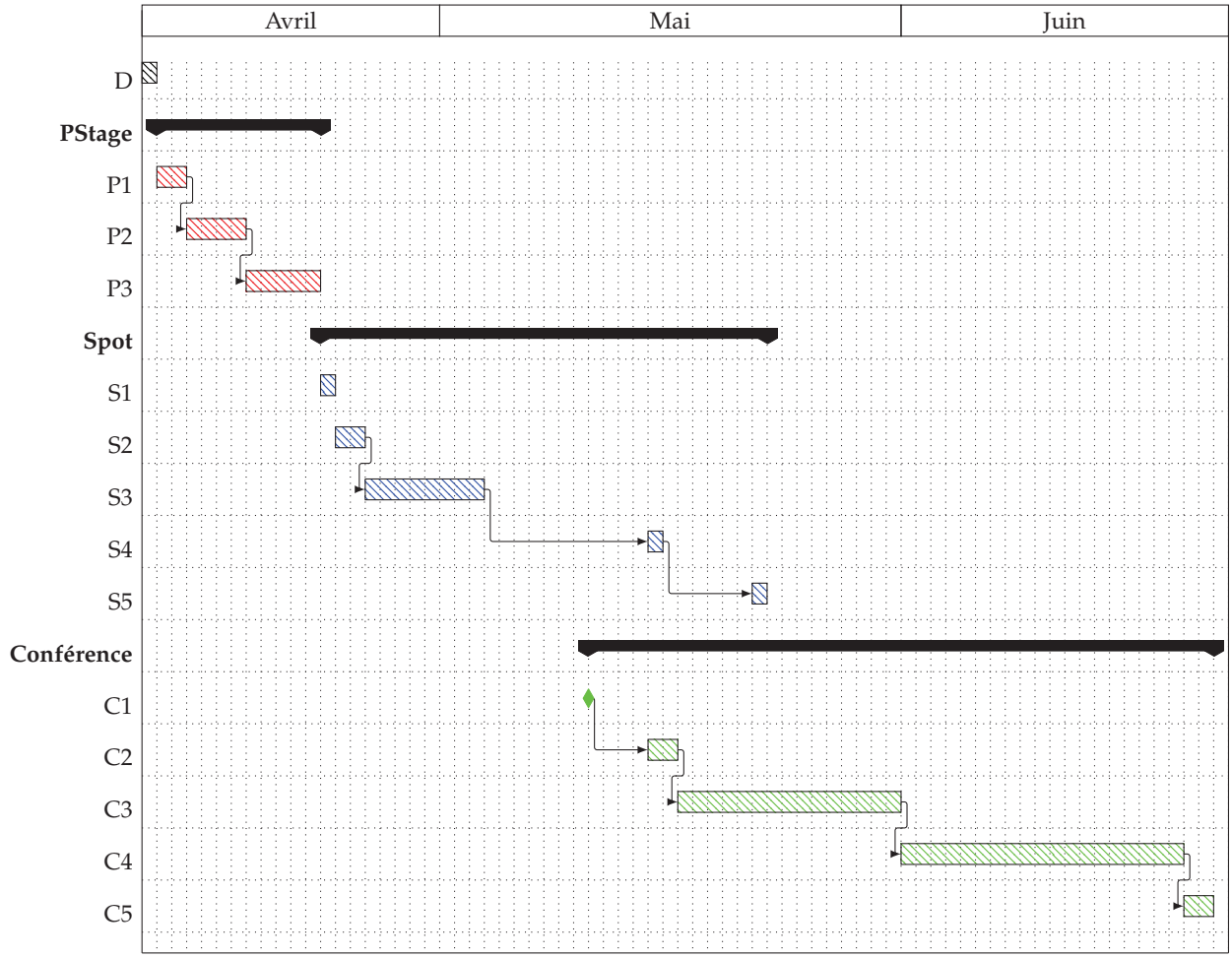
Le service informatique de Sciences Po Toulouse est constitué de James Ashton, responsable du service d'information, Patrick Piera et Philippe Bertholier.

Il a en charge la gestion du parc informatique et du réseau interne de l'IEP, des données présentes dans les bases de données et l'annuaire LDAP de l'établissement, ainsi que l'assistance aux utilisateurs.

1.2.2 Les collaborateurs et collaboratrices

J'ai été amené durant mon stage à travailler avec plusieurs collaborateurs et collaboratrices pour chacun des projets, je vais donc vous les présenter ci-dessous.

1. Fondation Nationale de Science politique, comprenant les huit Sciences Po de région et Sciences Po Paris.



Légende :

D Découverte de l'entreprise et installation de mon poste de travail.

P1 Recherche de documentation et familiarisation avec l'application PStage, destinée à gérer les conventions de stage des élèves de l'IEP.

P2 Installation et configuration de PStage-data, le module permettant la communication entre PStage et la base de données.

P3 Installation et configuration de PStage.

S1 Débuggage de l'application web Spot, gérant les convocations, listes d'appel et étiquettes des candidats au concours d'entrée de l'IEP.

S2 Familiarisation avec la base de données de l'application Spot et la celle recueillant les demandes d'inscription des candidats.

S3 Développement d'une application Java destinée à traiter puis transférer les données d'une base à l'autre.

S4 Transfert des données des candidats au concours d'entrée en deuxième année et modification des étiquettes.

S5 Transfert des données des candidats au concours d'entrée en quatrième année et modification de leur critère de tri.

C1 Réunion avec le service Comptabilité pour définir les types de paiement.

C2 Conception du MCD de la base de données et création de la base

C3 Conception de la partie de l'application destinée aux utilisateurs

C4 Conception de la partie de l'application destinée aux responsables d'ateliers.

C5 Tests de l'application.

FIGURE 1.1 – Diagramme de Gantt du stage

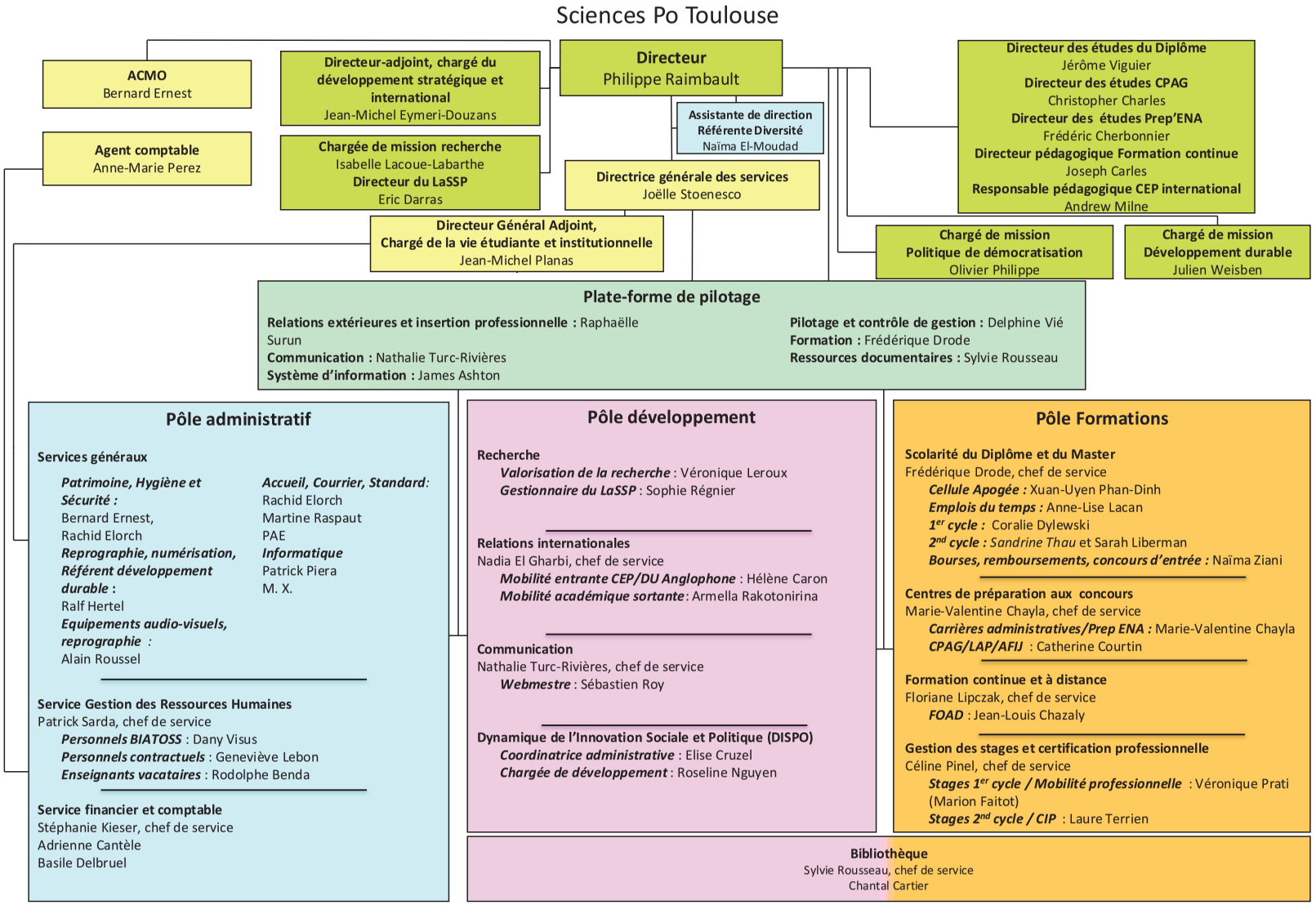


FIGURE 1.2 – Organisation administrative de Sciences Po Toulouse

James Ashton, responsable du système d'information

M. Ashton était mon maître de stage et mon principal interlocuteur durant ce stage. Il m'a présenté les outils utilisés à l'IEP avec lesquels je n'étais pas familière et m'a introduite auprès des autres collaborateurs et collaboratrices. C'était à lui que je présentais l'évolution de mon travail et à lui que je demandais de l'aide quand je rencontrais un problème technique.

Céline Pinel, responsable des stages et certification professionnelle

Mme Pinel est la personne ayant demandé la mise en place d'une application afin de simplifier la gestion de son service. Nous n'avons pas pu, pour des raisons techniques, terminer le projet PStage mais si cela avait été possible c'est elle qui m'aurait fourni les informations administratives nécessaires à la configuration de l'application.

Naïma Ziani, en charge des bourses, remboursements et concours d'entrée

Mme Ziani s'occupe des dossiers de candidatures aux concours d'entrée. J'ai principalement été en contact avec elle pour la gestion des concours d'entrée en deuxième année et quatrième année par l'intermédiaire de Spot. Elle m'a indiqué les informations à modifier sur les documents générés, ainsi que les modifications qu'elle souhaitait que j'apporte pour rendre Spot plus pratique à utiliser.

Véronique Leroux, responsable du Service Valorisation de la Recherche

Mme Leroux est organisatrice du Congrès Mondial ISA [2]/RCSL [3]. Elle m'a fournit les informations nécessaires à la réalisation du site d'inscription et de gestion du congrès.

2. International Sociological Association/Association Internationale de Sociologie
3. Research Committee on Sociology of Law/Comité de Recherche en Sociologie du Droit

Chapitre 2

PStage

2.1 Présentation de l'application

PStage est une application web en Java destinée à gérer les conventions de stage des étudiants et les offres d'emploi des entreprises, s'articulant autour d'une base de données MySQL [1], de l'annuaire LDAP [2] de l'établissement et d'Apogée [3]. Elle est développée par l'Université de Rennes depuis Avril 2011 en tant que projet d'ESUP-Portail, le consortium collaboratif créé en 2008 pour porter le projet national ENT "Espace Numérique de Travail".

PStage s'adresse donc à plusieurs publics :

- aux étudiants, qui peuvent ainsi chercher un établissement d'accueil ou une offre d'emploi ou de stage selon divers critères et créer leur convention,
- aux personnels administratifs, qui peuvent gérer les centres de gestion des stages ou les accords de partenariat,
- aux entreprises, qui sont libres de déposer des annonces et de gérer leurs contacts à leur guise.

Bien évidemment les privilèges des utilisateurs diffèrent selon leur statut, déterminés à l'aide des différentes sources de données de l'application. De plus l'application est largement paramétrable selon les désirs de l'établissement d'enseignement : validation des offres de stage par l'équipe administrative avant publication, envoi d'un email lors de l'ajout, la modification ou la suppression d'un établissement d'accueil ou autorisation pour les étudiants de modifier les coordonnées de l'entreprise lors de la création d'une convention, etc.

2.1.1 Outils utilisés

Afin de pouvoir administrer comme bon me semblait la machine qui allait héberger l'application j'ai créé une machine virtuelle, basée sur la technologie VMware – permettant d'émuler non seulement un BIOS mais aussi des périphériques tels que le lecteur de CD-ROM–, sur laquelle j'ai installé un système d'exploitation CentOS et un serveur web Apache.

CentOS étant le système d'exploitation utilisé sur les serveurs de l'IEP il m'a été imposée par mon maître de stage. Quand au serveur web, mon choix s'est porté sur Apache car il s'agit du serveur HTTP le plus utilisé, garantissant ainsi une documentation fiable et facile à trouver, et que j'avais déjà eu l'occasion de l'utiliser plusieurs fois au cours de ma formation à l'IUT.

PStage étant basés sur des servlets – classes Java permettant de créer dynamiquement des données au sein d'un serveur HTTP –, un moteur de servlet était nécessaire pour pouvoir exécuter l'application. J'ai opté pour Apache Tomcat, application libre possédant une grande communauté d'utilisateurs et donc très bien documenté. Bien évidemment, Tomcat étant une application développée en Java, j'ai installé une machine virtuelle Java avant.

1. Serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture.
2. Lightweight Directory Access Protocol, norme pour les systèmes d'annuaires : un annuaire LDAP est une structure arborescente dont chacun des nœuds est constitué d'attributs associés à leurs valeurs.
3. Application Pour l'Organisation et la Gestion des Enseignements et des Étudiants : logiciel développé par l'Agence de Mutualisation des Universités et des Établissements (AMUÉ) destiné à la gestion des inscriptions et des dossiers des étudiants dans les universités françaises.

2.2 Déploiement de PStagedata

PStagedata était le premier élément à déployer [4] sur le serveur Tomcat. En effet il s'agit du webservice [5] permettant à PStage de communiquer avec sa base de données.

2.2.1 Modification des fichiers de configuration

L'étape la plus longue est la configuration de l'application avant déploiement. En effet il faut renseigner :
- le répertoire de déploiement [6],
- le driver permettant la connexion à la base de données,
- l'url de la base de données,
- l'utilisateur ayant accès à la base de données,
- le mot de passe de l'utilisateur,
- le type d'API [7] que Java doit utiliser pour accéder à la base de données, JDBC [8] ou JNDI [9],
- le répertoire où stocker les fichiers temporaires,
- l'url de l'annuaire LDAP de l'établissement,
- l'url du serveur SMTP [10] de l'établissement,
- l'url du serveur Tomcat.

Cette étape est relativement peu intéressante, car il s'agit simplement de renseigner des données à l'application. Les seuls choix que j'ai eu à faire étaient les répertoire de déploiement et de stockage des fichiers temporaires, choix relativement peu cruciaux. Pour la base, le peu de documentation que j'avais réussi à me procurer n'évoquant que la méthode avec JDBC j'ai choisi cette option.

2.2.2 Mise en place de la base de données

L'étape suivante est la création et l'initialisation de la base de données.

Pour créer la base de données mySQL j'ai utilisé l'utilitaire phpMyAdmin, car il s'agissait de l'utilitaire utilisé par le service informatique de l'IEP. La création et le peuplement des tables est ensuite réalisée grâce à un script SQL fourni.

Le lancement de la tâche Ant [11] `init-data` permet ensuite d'indiquer à l'application que la base de données est accessible et prête à être utilisée.

2.2.3 Déploiement de l'application

Le déploiement de l'application se fait grâce à la tâche Ant `deploy`, qui compile les fichiers Java, place les classes compilées dans le répertoire webapp/WEB-INF/classes, respectant ainsi l'architecture d'un servlet Tomcat, et enfin copie ce répertoire dans le répertoire de déploiement de Tomcat sous le nom de esup-pstagedata.

Pour être sûr que le webservice fonctionne correctement je me suis rendue à l'adresse *http ://urlDeTomcat :8080/esup-pstagedata*. Le navigateur me retournait la page attendue, présentée dans la documentation, j'ai donc pu passer au déploiement de PStage.

2.3 Déploiement de PStage

2.3.1 Modification des fichiers de configuration

Là encore l'étape la plus longue était la configuration pré-déploiement. Pour PStage je devais renseigner :
- le répertoire de déploiement,
- l'url de la base de données,
- la clé Blowfish [12] générée par la tâche Ant `genBfKey`,

4. Fait de mettre en place un nouveau système ou un nouveau logiciel.
5. Programme informatique permettant la communication et l'échange de données entre applications dans des environnements distribués.
6. Le répertoire où les applications doivent être placées pour que Tomcat les déploie
7. Application Programming Interface ou interface de programmation : ensemble de fonctions, procédures ou classes permettant l'interaction des programmes les uns avec les autres.
8. Java DataBase Connectivity : API Java de connexion à des bases de données relationnelles
9. Java Naming and Directory Interface : API Java de connexion à des annuaires.
10. Simple Mail Transfer Protocol : protocole de communication utilisé pour transférer le courrier électronique vers les serveurs de messagerie électronique.
11. Projet de la fondation Apache utilisé pour automatiser la construction de projets, principalement des projets en langage Java
12. Algorithme de chiffrement conçu par Bruce Schneier en 1993, utilisant une clé de longueur variable (de 32 à 448 bits).

- la méthode d'authentification des administrateurs,
- le nom donné à l'application dans l'Espace Entreprise (utilisé notamment dans les mails),
- le nom donné à l'application dans l'Espace Stage,
- le code de l'université,
- le type de centres de gestion des conventions,
- plusieurs paramètres administratifs (autoriser ou non les centres de gestion à bloquer l'impression de la convention par les étudiants, autoriser ou non les entreprises à réserver les offres à certains centres de gestion, etc),
- le jour et mois du début de l'année universitaire,
- l'adresse web du webservice PStagedata,
- l'adresse web du serveur d'authentification,
- les noms d'utilisateurs des super-administrateurs (qui doivent être connus du CAS),
- les différentes adresses email de l'application (email de contact côté Entreprise et Stage mais aussi email expéditeur à indiquer au serveur SMTP),
- l'adresse web du serveur SMTP,
- le répertoire où stocker les fichiers temporaires,
- les répertoire où stocker les fichiers téléchargés sur le serveur (logos d'entreprises et de centres de gestion, fichiers attachés aux offres),
- les paramètres de téléchargement de ces fichiers (tailles maximales et extensions autorisées pour les fichiers et les images),
- le répertoire contenant les fichiers xml et xsl servant à générer les conventions,
- les paramètres d'accès à l'annuaire LDAP (adresse web, nom d'utilisateur et mot de passe),
- les valeurs de l'attribut ldap.affiliation pour un employé, un prof et un étudiant dans l'annuaire LDAP (utilisé pour déterminer des privilèges de l'utilisateur dans l'application),
- les différents attributs LDAP stockant les informations d'une personne,
- les types de personnel ne pouvant pas être tuteur de stage,
- la configuration choisie pour récupérer les données des utilisateurs.

La plupart de ces valeurs étaient destinées à des configurations administratives et n'avaient donc que peu d'intérêt pour moi, puisque mon but était de réussir à avoir une application déployée fonctionnelle, la configuration administrative venant après. Les seuls choix déterminants pour le fonctionnement de l'application étaient ceux listés ci-après.

La méthode d'authentification des administrateurs : J'avais le choix entre le CAS [13] ou Shibboleth [14]. Or le système d'authentification utilisé pour l'Université Toulouse 1 est le CAS, je n'ai donc pas eu le choix sur ce point.

Le type des centres de gestion des conventions : PStage offre plusieurs options pour gérer les centres de gestion :

- avoir un centre établissement dont dépendent des UFR,
- avoir un centre établissement dont dépendent des étapes,
- avoir un centre établissement dont dépendent des UFR et des étapes,
- avoir seulement un centre établissement gérant toutes les conventions.

Dans notre cas, le critère le plus conforme est le dernier puisque toutes les conventions seront gérées par le service des stages de l'IEP.

Les adresses web du webservice PStagedata, du serveur d'authentification (le serveur CAS dans notre cas donc) et les paramètres d'accès à l'annuaire LDAP : Les informations concernant le serveur CAS et l'annuaire LDAP m'ont été fournies par M. Ashton. Pour ce qui est de PStagedata, l'adresse web du webservice pouvait être trouvée à l'adresse servant à valider le fonctionnement du webservice (indiquée au 3.2.3).

Les différents attributs LDAP stockant les informations d'une personne : Dans mon cas les valeurs par défaut étaient les bonnes mais il était nécessaire de vérifier que l'architecture de l'annuaire LDAP utilisé corresponde bien à l'architecture par défaut du fichier de configuration.

13. Central Authentication Service : système d'authentification unique pour le web
14. Mécanisme de propagation d'identités, développé par le consortium Internet2, qui regroupe 207 universités et centres de recherches.

La configuration choisie pour récupérer les données des utilisateurs : Ce point est le plus crucial de la configuration car il détermine la ou les source(s) de données de l'application. Celle-ci dispose de trois configurations :

- utiliser les webservice esup-si communiquant avec Apogée et LDAP en association,
- passer uniquement par LDAP,
- développer un webservice propre à l'université.

Nous n'avions ni le temps ni les besoins de mettre en place la troisième solution et la première présentait un problème de taille : l'utilisation des webservices esup-si impliquait l'installation d'un patch [15], le patch AMUE, sur Apogée, or nous n'avions pas les moyens de faire cette installation car il était administré par l'Université Capitole. Nous avons donc opté pour un accès tout LDAP.

2.3.2 Déploiement de l'application

Le déploiement de l'application s'est fait de manière identique à celui de PStagedata.

Cependant, après quelques jours de débogage et l'aide des autres utilisateurs de PStage, je me suis rendue compte que pour utiliser une configuration exclusivement basée sur LDAP il était nécessaire que l'annuaire ait une architecture spécifique, à savoir qu'il comporte une branche contenant les différentes composantes de l'établissement. Or ce n'était pas le cas pour celui de l'IEP, et il était impossible de le modifier.

Il n'était donc pas possible de configurer PStage avec notre annuaire LDAP seul.

Étant donné que l'Université Capitole utilise également PStage avec une configuration LDAP + Apogée, nous avons demandé à pouvoir utiliser leurs webservices mais nous n'avons toujours pas eu de réponse à l'heure où j'écris ces lignes.

2.4 Bilan du projet

J'aurais pour la première fois lors de ce projet travaillé avec des servlets. Malheureusement, étant donné la complexité de l'application, je n'ai pas pu bien comprendre et appréhender cette nouvelle syntaxe. À ce titre j'ai énormément regretté l'absence de documentation du projet, le seul document que j'ai réussi à trouver étant celui décrivant les différentes étapes du déploiement. Je pense que ce manque de documentation m'a handicapée, en effet ne pouvant comprendre totalement le fonctionnement de l'application j'ai parfois du chercher les causes d'erreur à tâtons, méthode relativement lente et peu fiable.

Malgré cela j'ai trouvé intéressante cette expérience, en effet je n'avais jamais eu jusqu'à présent à mettre en place une application qui m'était totalement inconnue et j'ai pu réaliser que la tâche était bien moins aisée que ce que l'on pourrait penser au premier abord.

De plus j'ai découvert CentOS que j'ai trouvé stable et possédant des paquets plus récents que les distributions avec lesquelles j'étais familière, telles que Debian.

Mon impression sur les annuaires LDAP est partagée. Je trouve ce système simple et léger mais également peu évident à manipuler et possédant un langage de recherche peu intelligible.

15. Section de code que l'on ajoute à un logiciel, pour y apporter des modifications.

Chapitre 3

Spot

3.1 Présentation de l'application

Spot est une application web développée il y a plusieurs années pour simplifier la gestion des concours de l'IEP. Elle permet de :

- saisir les dossiers des candidats,
- générer les convocations,
- générer les listes d'appel,
- générer les étiquettes d'anonymat,
- générer les étiquettes de tables,
- saisir les notes,
- générer les procès verbaux de délibération vierges,
- valider les résultats,
- effectuer des statistiques sur différents critères (sexe des candidats, série ou mention du bac, nationalité, département, etc).

Cependant, aujourd'hui, les candidats s'inscrivent directement au concours sur le site web de l'IEP et les dossiers sont validés par le personnel administratif grâce à un formulaire.

Nous avons donc deux bases de données : celle alimentant Spot et celle recueillant les données des formulaires d'inscription et de validation. Ma tâche était de trouver un moyen, une fois les inscriptions terminées, de transférer les données des candidats d'une base de données à l'autre afin de pouvoir générer convocations, listes d'appel et étiquettes d'anonymat et de table.

3.2 Analyse des bases de données

3.2.1 Compréhension des interactions entre les différentes tables de la base `spot`

Afin d'identifier où et sous quelle forme insérer les données depuis les tables `2a` et `4a`, contenant respectivement les candidats aux concours d'entrée en deuxième et quatrième année, de la base `inscription` je devais d'abord comprendre comment étaient organisées les tables de `spot` les unes par rapport aux autres. Étant donné que les tables ne disposaient d'aucune clés étrangères pour les relier entre elles, il a fallu que j'insère des "candidats tests" dans spot pour comprendre comment elles interagissaient. Étant donné que pour générer les convocations et les étiquettes seules les informations d'identité et d'adresse étaient indispensables, je me suis concentrée sur celles-ci. J'en suis arrivé au MCD présenté dans la *figure 4.1*.

3.2.2 Correspondance entre les données d'`inscription` et de `spot`

Grâce aux candidats-tests insérés dans `spot` et aux formulaires d'inscription j'ai pu réaliser un tableau de correspondance des données entre les deux bases.

Certaines informations n'ont cependant pas été prises en compte car elles n'avaient pas leur place dans `spot` ou qu'elle n'avaient pas d'intérêt pour la génération des convocations et étiquettes.

Arès avoir identifié où j'allais devoir insérer chaque donnée, il a fallu que je vérifie qu'il n'y aurait pas de problème de compatibilité en passant d'une base à l'autre. C'est ainsi que je me suis rendue compte qu'il me faudrait modifier certaines colonnes afin qu'elles soient compatibles avec `spot`.

`id` : Les identifiants des données des tables `2a` et `4a` d'inscription étaient des entiers générés par auto-incrément et démarrant chacun à 1. Or si l'on laissait les identifiants comme ceci la plupart se retrouvaient

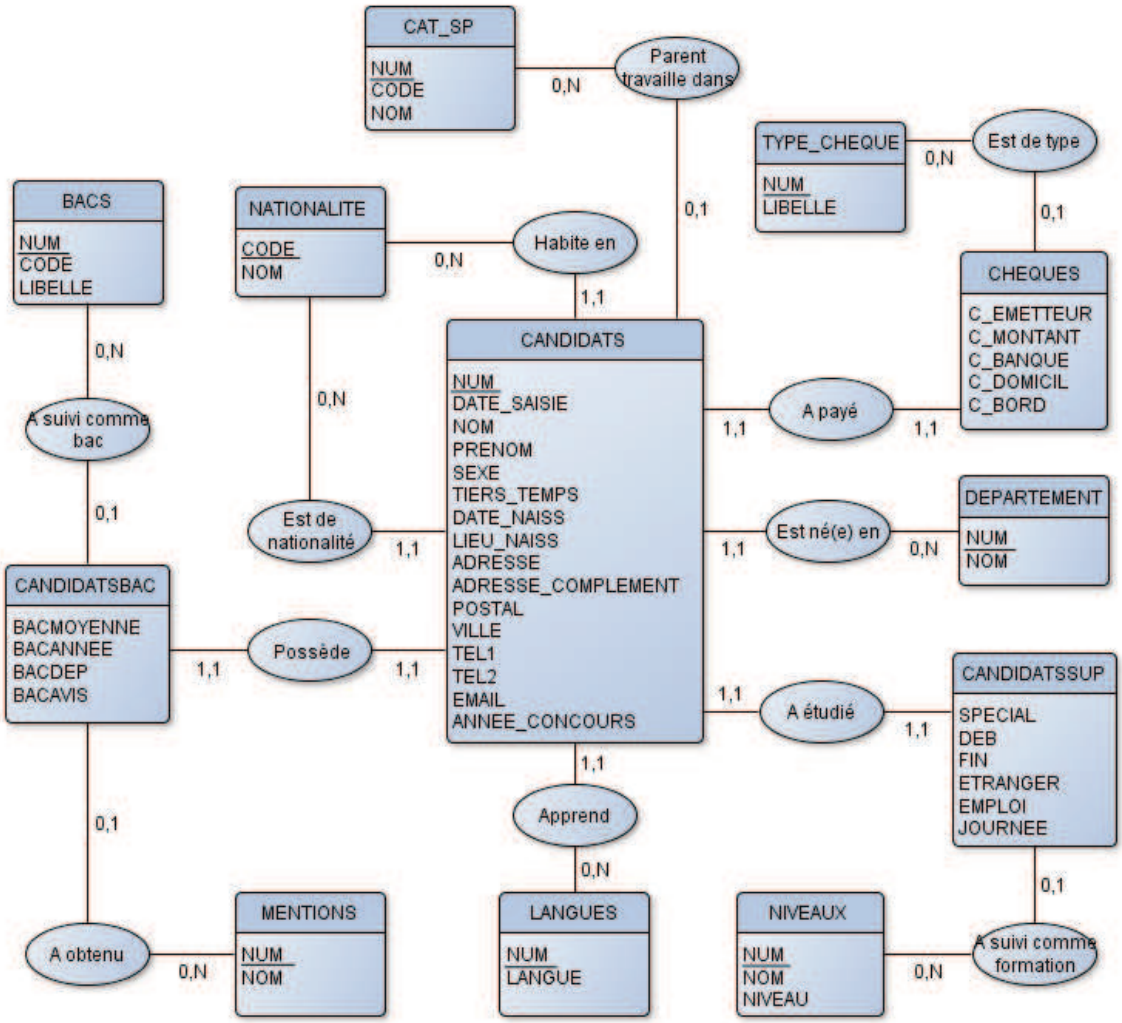


FIGURE 3.1 – MCD d'une partie de la base `spot`

en doublon lors de l'insertion dans `spot` car les candidats au concours d'entrée en deuxième et quatrième année étaient "mélangés" dans les tables. Il me fallait donc trouver un moyen de rendre chaque identifiant unique.

`adresse` : Dans `inscription`, suite à une erreur de conception corrigée depuis, l'adresse était entièrement stockée dans le champ `adresse` pour de nombreux candidats. Or dans `spot` il était nécessaire de disposer distinctement des éléments `ADRESSE`, `POSTAL` et `VILLE`.

`sexe, tiers_tps et boursier` : Ces informations n'étaient pas stockées sous le même format dans les deux bases. Il fallait donc les transformer avant de les insérer afin qu'elles soient exploitables pour `Spot`.

`nationalite, pays, mention, langue, serie, nature_ects et dep` : Ces champs étaient remplacés dans spot par une clé étrangère. Il fallait donc aller chercher la clé correspondante dans les tables `NATIONALITE`, `MENTIONS`, `LANGUES`, `BACS`, `NIVEAUX` et `DEPARTEMENTS`.

3.3 Conception et développement de `Spotted`

3.3.1 Choix du JDBC

J'ai tout d'abord souhaité faire une procédure SQL mais je me suis rendu compte que le découpage des adresses risquait de présenter une certaine difficulté.

J'ai donc fait le choix de développer mon application, `Spotted`, en Java. En effet ce langage offre des outils simples et efficaces pour gérer les bases de données, mais aussi pour traiter des chaînes de caractères à l'aide d'expressions régulières [1], outil indispensable pour découper simplement les adresses.

1. Motif décrivant un ensemble de chaînes de caractères possibles selon une syntaxe précise.

De plus j'avais déjà une expérience avec le JDBC grâce au module «Base de Données Avancée» suivi durant le quatrième semestre de ma formation.

3.3.2 Conception de l'application

À partir de mes cours de Base de Données Avancées, j'ai pensé que le plus simple pour modifier les données serait de créer un curseur éditable afin de modifier les données à la volée avant de réaliser les insertions dans les tables de `spot`. Mais je ne voulais pas éditer les données originelles d'`inscription`, en effet en cas d'erreur les données auraient alors été irrécupérables. J'ai donc décidé de créer deux tables «temporaires» ayant la même structure que, respectivement, `2a` et `4a` servant à stocker les données des candidats le temps de la modification.

L'utilisateur pouvait vouloir traiter les candidats de deuxième ou de quatrième année. Il était donc nécessaire de prévoir un moyen pour celui-ci d'indiquer son choix au programme. De plus, afin d'éviter d'avoir le nom d'utilisateur et le mot de passe d'accès à la base, il fallait aussi prévoir sa demande à l'utilisateur.

Ainsi après analyse le fonctionnement de mon application pouvait se résumer comme suit :

- demander quelle table traiter,
- demander s'il faut initialiser les tables de spot,
- demander les identifiants pour accéder à la base,
- se connecter à la base,
- copier le contenu de la base à traiter dans sa table temporaire après avoir vidé celle-ci,
- effectuer les traitements requis,
- insérer les données traitées dans `spot`,
- se déconnecter.

Ainsi j'ai décidé de réaliser une application répondant au schéma suivant :

- une classe gérant les interactions avec les bases `spot` et `inscription` : connexion, déconnexion et requêtes,
- une classe ayant pour rôle de générer les requêtes nécessaires au programme et de récupérer les données fournies par ces requêtes,
- une classe spécialisant la précédente et réalisant tous les traitements nécessaires à l'insertion des données des candidats dans la base `spot`,
- une classe servant d'interface avec l'utilisateur, lui permettant de choisir la table à traiter, s'il souhaite initialiser spot, et de fournir ses informations d'authentification.

La figure 4.2 présente le diagramme de classe de l'application.

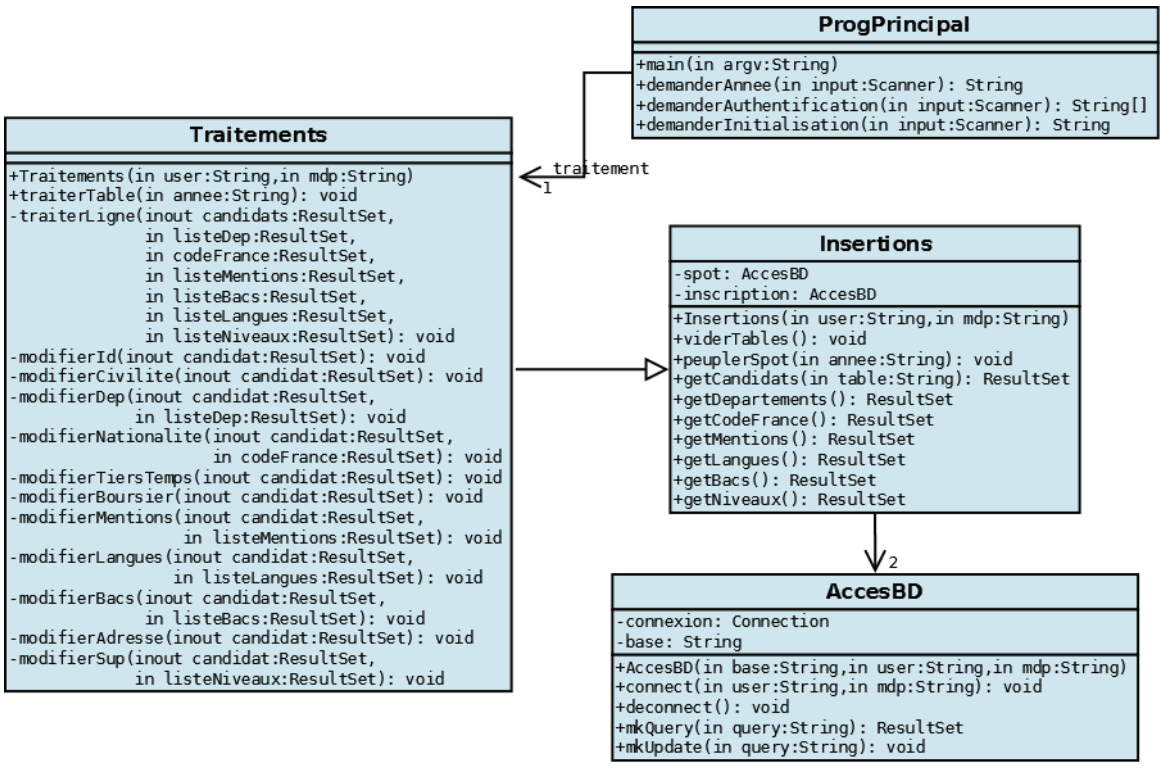


FIGURE 3.2 – Diagramme de classe de `Spotted`

3.3.3 Développement de l'application

Durant le développement de l'application je me suis rendu compte de certains oublis de conception. Je n'avais notamment aucun moyen de récupérer les objets Statement créés durant l'exécution du programme afin de les fermer proprement avant la déconnexion de la base. Pour remédier à ce problème j'ai ajouté à la classe gérant les interactions avec les bases un attribut de type liste. À chaque fois qu'un objet Statement était créé, il était rajouté à cette liste et avant la déconnexion je n'avais qu'à parcourir cette liste pour fermer tous les Statements.

J'ai également réalisé qu'il était beaucoup plus simple de récupérer une donnée dans une liste ou un tableau associatif que dans un curseur. Au moment de la refactorisation j'ai donc fait en sorte que la classe récupérant les données renvoie non plus des ResultSet mais des listes ou des tableaux associatif, exception faite bien entendu des données des candidats à traiter, ces données étant toujours dans un curseur éditable.

3.4 Modifications de `Spot`

Après la migration des données je me suis rendue compte d'un oubli dans ma conception : la gestion des candidats en doublon. Heureusement ils étaient peu nombreux, nous avons donc pu, Mme Ziani et moi-même, les traiter manuellement afin de ne pas perdre de temps

Il a ensuite fallu que j'apporte de menues modifications à Spot.

Génération des étiquettes

Les étiquettes générés par Spot ne correspondaient pas au format d'étiquettes que possédait Mme Ziani, il a donc fallu que je modifie la mise en page de celles-ci.

Les documents étant générés à l'aide de la classe PHP libre FPDF, par souci de cohérence et d'uniformité, j'ai moi aussi utilisé FPDF pour la génération des nouvelles étiquettes. En effet je trouvais inutile et illogique d'utiliser dans une même application plusieurs méthodes différentes pour générer des documents au format PDF.

Génération des documents des candidats au concours d'entrée en quatrième année

La page permettant de générer les listes d'appel et étiquettes triait les candidats selon leur LV2 (Espagnol, Allemand ou Italien), or Mme Ziani souhaitait répartir les candidats au concours d'entrée en quatrième année selon la spécialité qu'ils avaient choisi pour leur dissertation (Histoire, Sciences politiques, Économie ou Droit).

Or le choix fait par les candidats lors de l'inscription n'était pas présent dans Spot. Pour remédier à ce problème j'ai ajouté à la fin de la table `CANDIDATS`, afin d'éviter tout éventuel problème d'insertion (dans le cas où le nom des colonnes n'aurait pas été précisé dans la requête), une colonne stockant cette information. Après cela j'ai modifié `Spotted` afin de prendre en compte cette modification, puis ai retransféré les données des candidats de 4a, avec leur spécialité cette fois-ci.

Par la suite j'ai simplement ajouté une condition sur l'année traitée dans le script php permettant de générer les documents afin que, si l'année demandée par l'utilisateur était la 4, les résultats retournés soient ordonnés selon la spécialité et non selon la langue des candidats.

3.5 Bilan du projet

J'ai beaucoup aimé travailler sur Spot, j'ai pu réaliser à quel point une base de données «mal» réalisée, sans aucune gestion des clés étrangères et avec des types de colonnes pas toujours adaptés, rendait complexe la compréhension et la gestion de celle-ci. De plus j'ai pu mettre en pratique mes connaissances en JDBC.

Cela m'a conforté dans mon souhait de me spécialiser dans la création et gestion de bases de données.

Chapitre 4

Congrès RCSL 2013

4.1 Présentation du projet

L'IEP de Toulouse organise en collaboration avec le Research Committee on Sociology of Law (RCSL), l'un des cinquante-trois comités de recherche de l'Association Internationale de Sociologie, un congrès qui aura lieu à Toulouse, du 3 au 6 septembre 2013, sur le thème : Sociologie du Droit et Action Politique.

Ma tâche était de créer :

- une page web permettant aux congressistes de proposer des communications,
- une page permettant aux congressistes de s'inscrire au congrès,
- une page permettant aux responsables d'ateliers de gérer les communications proposées,
- une page permettant aux organisateurs de gérer les inscriptions.

Je n'ai pas eu à réaliser d'audit car Véronique Leroux, organisatrice de ce congrès, m'a fourni un document contenant toutes les informations nécessaires au développement. Je n'ai eu qu'à clarifier le document pour supprimer les quelques imprécisions présentes.

Précisions sur la gestion des inscriptions : Les inscriptions au congrès peuvent se faire de deux manières différentes.

Si le congressiste paye son inscription par carte bancaire son inscription est immédiate et effective.

Dans le cas contraire (paiement par virement ou mandat international) il est seulement préinscit et son inscription sera définitivement prise en compte une fois son paiement confirmé aux organisateurs du congrès par le service Comptabilité de l'IEP.

4.1.1 Présentation de Symfony

Ma principale contrainte lors de ce projet était l'utilisation du framework php Symfony 2.0, car la plupart des applications «grand public» de l'IEP avaient été développées à l'aide de Symfony1. Il paraissait donc logique de garder cette démarche pour faciliter la maintenance. Cependant la question se posait de développer avec la même version du framework que les applications existantes ou de passer à la version 2, non compatibles avec les versions 1.x. Mais étant donné que les versions 1.x ne sont plus soutenues, il a été décidé que je coderai sous Symfony2.

Un framework est un ensemble d'outils et de composants logiciels organisés conformément à un plan d'architecture et des patrons de conception, l'ensemble formant un squelette de programme. Ils simplifient ainsi la maintenance des programmes et favorise la réutilisation du code.

Symfony est un des frameworks PHP les plus reconnus aujourd'hui. Il est construit selon le modèle MVC (présenté plus en détail dans la *section 4.4.1*), permettant une grande maintenabilité et évolutivité du code, gère nativement les url parlantes – mécanisme permettant de formater l'url d'une page indépendamment de sa position dans l'arborescence fonctionnelle, et ainsi d'optimiser le référencement de celle-ci – et un ORM, Doctrine (présenté dans la *section 4.3.1*), permettant de gérer simplement les données de l'application. En outre il facilite la gestion de l'IHM en intégrant un moteur de template [1], Twig, sécurisé, flexible et très facile à prendre en main.

1. Modèle de document dans lequel il ne reste plus qu'à introduire un contenu spécifique.

4.2 Création du bundle

Symfony est basé sur le concept de «bundle». Il s'agit de regrouper en un même endroits les éléments traitant une même fonctionnalité, telle une «brique» du logiciel. Cela permet d'obtenir un code plus propre mais également de favoriser la réutilisation de celui-ci, puisque étant donné que le code est regroupé, il est aisé de l'exporter vers une autre application.

Il fallait donc commencer par créer notre «brique» (étant donné la faible taille de l'application j'ai trouvé plus pertinent de ne réaliser qu'un seul bundle mais il est tout à fait possible, voire nécessaire, de créer plusieurs bundles dans une application plus conséquente). Cette création peut se faire manuellement mais Symfony dispose de nombreux scripts accessibles en ligne de commande générant automatiquement le code de base, dont un générateur de bundle qui permet facilement de créer toute l'arborescence dont nous auront besoin par la suite et d'enregistrer le nouveau bundle dans le noyau de l'application, afin que celle-ci sache qu'il existe et où elle peut le trouver.

4.3 Création de la base de données avec Doctrine

À partir du document fourni par Mme Leroux, j'ai réalisé le MCD présent sur la *figure 4.1* pour la base de données de l'application.

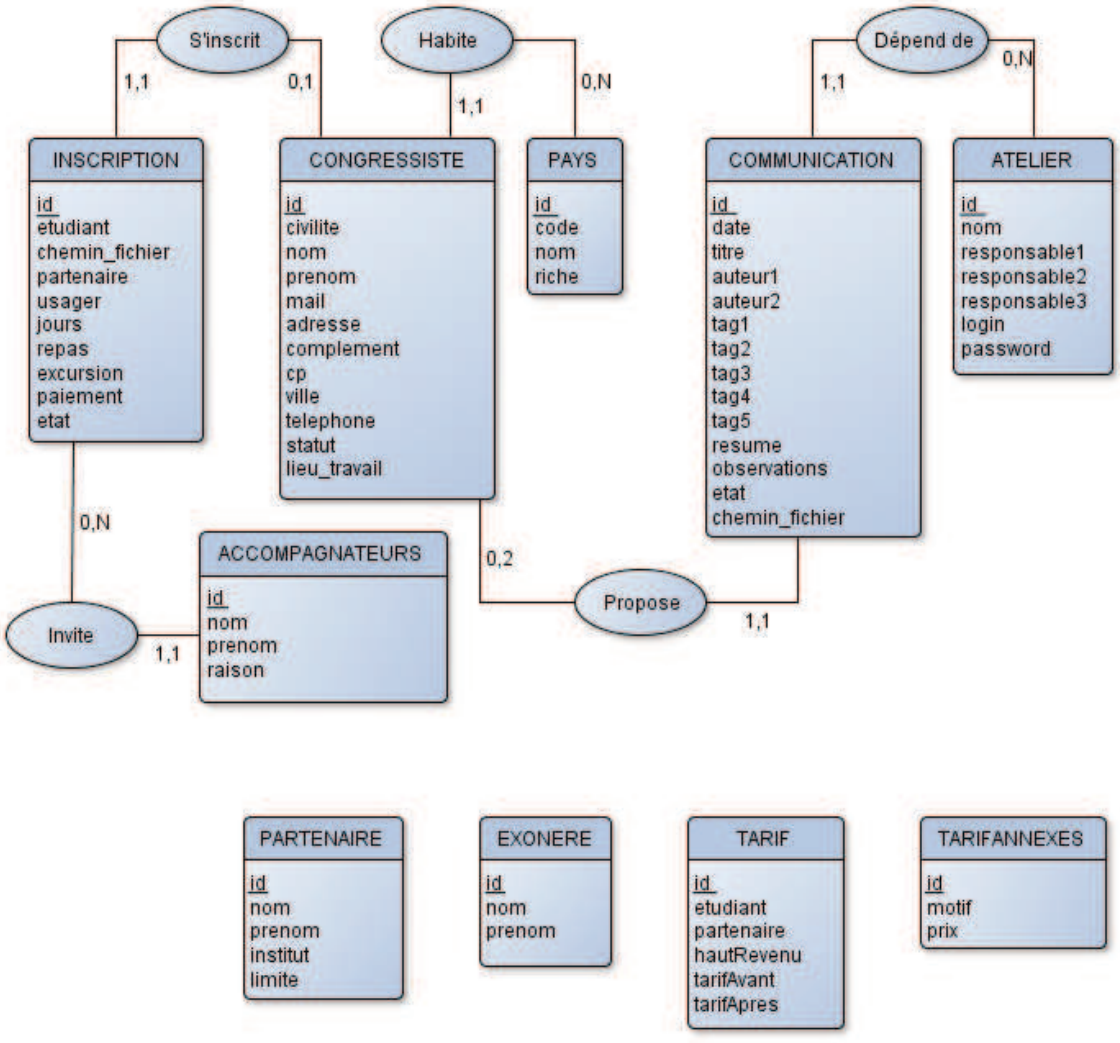


FIGURE 4.1 – MCD de la base alimentant l'application d'inscription et de gestion du Congrès RCSL 2013

Afin de créer cette base j'ai utilisé Doctrine, l'ORM par défaut de Symfony.

4.3.1 Qu'est-ce que Doctrine ?

Doctrine est un ORM, ou Object-Relational Mapping (Mapping Objet-Relationnel en français). Il s'agit d'un ensemble de classes permettant de manipuler une base de données relationnelle comme une base de données orientée objet en définissant des correspondances entre les tuples de la base de données et les objets du langage utilisé.

Doctrine est l'un des ORM php les plus connus et utilisés. Il est actuellement l'ORM par défaut de nombreux frameworks, dont Symfony depuis la version 1.3.

Il possède son propre langage de requête, le Doctrine Query Language (DQL) faisant le lien entre un langage objet et le langage SQL.

4.3.2 Création des entités

Nous avons vu précédemment que Doctrine faisait le lien entre des objets php et les éléments présents dans la base de données. La première étape était donc de créer ces objets, appelés `Entités`. Symfony possède un script générateur d'entité qui gère automatiquement l'identifiant et auquel il faut simplement passer le nom du bundle dans lequel on souhaite créer notre entité puis les noms et types des attributs de l'objet (il s'agit là des types Doctrine – array, object, boolean, integer, smallint, bigint, string, text, datetime, datetimetz, date, time, decimal, float – et non pas SQL, Doctrine agissant comme une boîte noire gérant totalement l'aspect SQL de la base de données). On obtient alors une classe portant le nom de notre entité et comportant, pour chacun des attributs passé au générateur, l'attribut - dont la correspondance avec la base est assuré grâce à une annotation Doctrine - et ses accesseurs en consultation et modification. La gestion des clés étrangères doit par contre se faire à la main.

Exemple d'attributs gérés par Doctrine grâce aux annotations

```
<?php
use Doctrine\ORM\Mapping as ORM;
/**
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="inscription\VisiteurBundle\Entity\AccompagnateurRepository")
 */
class Accompagnateur
{
    /**
     * Clé primaire de la table Accompagnateur
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * Clé étrangère 1..N faisant le lien entre les tables Accompagnateur et Inscription
     * @ORM\ManyToOne(
          targetEntity="inscription\VisiteurBundle\Entity\Inscription",
          inversedBy="accompagnateurs"
       )
     */
    private $inscription;

    /**
     * @ORM\Column(name="nom", type="string", length=30, nullable=false)
     */
    private $nom;
...
}
?>
```

4.3.3 Validation des données

Symfony permet de mettre en place très simplement des critères de validation des données avant toute insertion dans la base. Il s'agit non seulement d'une sécurité mais aussi d'un confort indispensable pour les utilisateurs ayant à utiliser le formulaire collectant les données.

Ainsi si l'utilisateur ne rentre rien dans un champ du formulaire alors que le champ correspondant dans la table SQL possède une contrainte de type `not null`, au moment de la validation au lieu de voir s'afficher une page d'erreur incompréhensible renvoyée par MySQL l'utilisateur sera redirigé vers le formulaire avec un message lui indiquant qu'il doit obligatoirement fournir une valeur.

Exemple de règles de validation définies par annotations

```
<?php
use Symfony\Component\Validator\Constraints as Assert;
...
class Accompagnateur
{
    ...
    /**
     * @Assert\NotBlank(
     *          message =   "Les noms de vos accompagnateurs doivent être renseignés."
     * )
     * @Assert\MaxLength(
     *          limit   =   30,
     *          message =   "Nom trop long (30 caractères maximum)."
     * )
     * @ORM\Column(name="nom", type="string", length=30, nullable=false)
     */
    private $nom;
    ...
    /**
     * @Assert\Choice(
     *          choices =   {"repas", "excur", "both"},
     *          message =   "Type de raison invalide."
     * )
     * @ORM\Column(name="raison", type="string", length=5, nullable=false)
     */
    private $raison;
    ...
}
?>
```

4.4 Architecture de l'application

4.4.1 Définition du MVC

Le MVC, ou Modèle-Vue-Contrôleur, est une méthode de conception imposant la séparation entre les données, la présentation de ces données et les traitements. Symfony utilise ce modèle pour une plus grande maintenabilité et évolutivité.

Le modèle assure la gestion des données et garantit leur intégrité. Il offre des méthodes pour les mettre à jour et les récupérer. Il s'agit pour nous de la base de données MySQL ainsi que des entités créées précédemment.

La vue correspond à l'interface avec laquelle l'utilisateur interagit. Dans notre cas les vues étaient conçues grâce à Twig.

Le contrôleur prend en charge la gestion des événements : il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer selon celles-ci.

4.4.2 Diagramme de l'application

Je n'ai pas trouvé pertinent de réaliser un diagramme de classe étant donné l'omniprésence de Symfony. En effet un tel diagramme aurait complètement «noyé» l'architecture de l'application dans l'architecture du fra-

mework. J'ai donc préféré présenter dans la *figure 4.2* un schéma simplifié illustrant les différentes interactions entres les contrôleurs, modèles et vues de ce site.

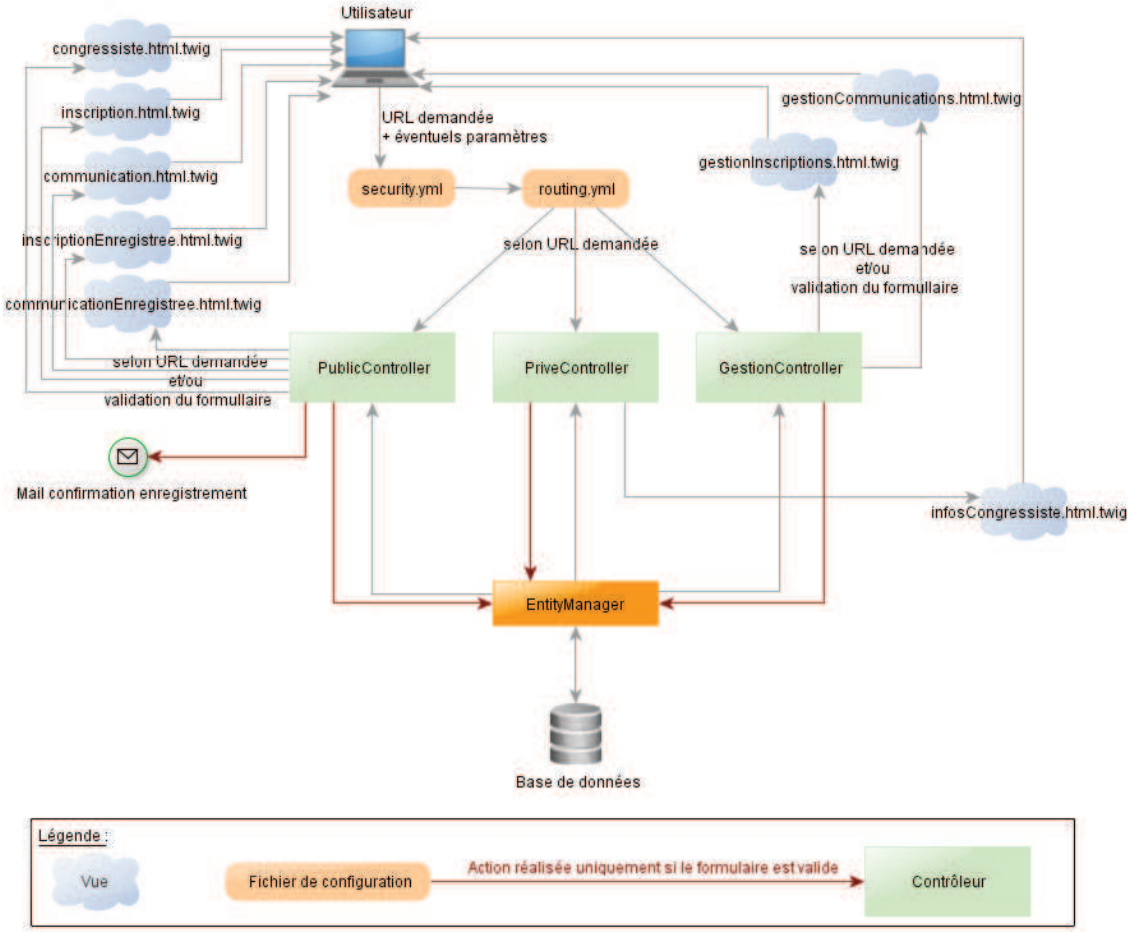


FIGURE 4.2 – Diagramme MVC du site d'inscription et de gestion du Congrès RCSL 2013

routing.yml

Ce fichier fait le lien entre les URL demandées et les contrôleurs à appeler. La route définit ainsi :

```
inscription_visiteur_accueil:
   pattern:  /{locale}/inscription
   defaults: { _controller: inscriptionVisiteurBundle:Formulaire:accueil, locale: fr }
   requirements:
       locale: fr|en
```

peut se traduire comme cela :
si l'url demandée est «urlDuSite/fr/inscription» ou «urlDuSite/en/inscription» appeler la méthode `accueilAction` du contrôleur `FormulaireController` présent dans le bundle `inscriptionVisiteurBundle`. Si aucune valeur n'est précisée pour `locale` prendre «fr» par défaut.

security.yml

Ce fichier permet à Symfony de gérer les utilisateurs par l'intermédiaire des pare-feux. Un pare-feu est un mécanisme permettant de sécuriser une partie de l'application selon sa route d'accès. Si un utilisateur anonyme essaie d'accéder à une ressource protégée le pare-feu le redirige automatiquement vers une demande d'authentification. De même c'est le pare-feu qui va intervenir si un utilisateur identifié tente de voir une donnée mais qu'il n'a pas suffisamment de privilèges pour cela.

Voici par exemple la définition du pare-feu protégeant l'accès aux vues permettant de gérer le congrès (il ne s'agit bien évidemment ici que d'extraits de configuration et non du fichier. complet) :

```
security:
    providers:
        gestion:
          entity: { class: inscriptionVisiteurBundle:Atelier, property: login }

    firewalls:
        gestion:
            pattern: .*/gestion.*
            form_login:
                check_path: /gestion_check
                login_path: /login_g
                default_target_path: gestion_ateliers

    access_control:
        - { path: /gestion, role: ROLE_ADMIN }
        - { path: /.*, role: IS_AUTHENTICATED_ANONYMOUSLY }
```

`providers` indique à Symfony où il doit chercher les utilisateurs. Il peut s'agir d'une entité, d'un fichier externe ou encore d'utilisateurs codés en dur dans le fichier. Dans notre cas la source d'utilisateurs est la classe `Atelier`, et le nom d'utilisateur est l'attribut `login`. Il faut noter que les entités servant de source d'utilisateurs doivent implémenter l'interface `UserInterface`.

`firewalls` permet de définir les pares-feu.

- `pattern` permet d'indiquer à Symfony les routes gérés par ce pare-feu (ces routes sont définies par des expressions régulières),
- `login_path` indique la route de la page d'authentification,
- `check_path` indique la route de vérification de l'authentification (cette vérification est traitée automatiquement par Symfony, il n'y a pas de contrôleur à implémenter),
- `default_target_path` indique la page à renvoyer à l'utilisateur s'il est arrivé directement sur la page d'authentification (s'il a été redirigé sur la page d'authentification après avoir demandé une page il sera redirigé vers celle-ci une fois l'authentification validée),

`access_control` permet de préciser les rôles nécessaires aux utilisateurs afin d'accéder aux données.

PublicController

Ce contrôleur gère toutes les pages ne nécessitant aucune authentification, telles que les pages contenant les formulaires d'inscription et de proposition de communication.

PriveController

Ce contrôleur gère la page permettant à un congressiste ayant proposé au moins une communication de modifier ses informations. Elle nécessite une authentification, qui consiste en l'adresse mail du congressiste en tant qu'identifiant et «RCSL» suivi de son identifiant dans la base de données comme mot de passe.

GestionController

Ce contrôleur gère les vues des responsables d'ateliers, leur permettant de voir les différentes communications proposées pour leur atelier et de les accepter ou les refuser. Si l'utilisateur authentifié possède le rôle de superutilisateur une vue lui permet également de voir les communications non encore traitées des autres ateliers et de gérer les préinscriptions.

4.5 Récupération des données

La composante la plus intéressante de la réalisation de ce projet a été pour moi la récupération des données à l'aide de Doctrine.

Dans Symfony toutes les méthodes de récupération de données sont regroupées dans un «repository». Il y a un repository par entité.

La récupération «basique» des données est rendue très simple par les méthodes de base Doctrine.

```
$repo = $this->getDoctrine()
             ->getEntityManager()
             ->getRepository('inscriptionVisiteurBundle:Communication');
```

```
//retourne tous les éléments présents dans la table Communication
$repo->findAll();
//retourne l\rq{}élément de la table Communication ayant comme clé primaire $id
$repo->find($id);
//retourne les éléments de la table Communication ayant comme premier auteur $auteur1
$repo->findBy(array('auteur1' => $auteur1));
```

Ces requêtes renvoient un résultat sous forme d'un tableau d'objets.

Méthodes de récupération personnalisées

Lorsque les informations à récupérer nécessitaient l'utilisation d'une requête plus complexe j'ai eu à coder des méthodes de récupération personnelles. Ces requêtes étaient implémentées dans le repository de l'entité à récupérer, afin de garder le code le plus clair et compréhensible possible.

Ces requêtes sont construites à l'aide du QueryBuilder, un outil permettant de créer une requête de toute pièce en spécifiant nos conditions.

Ainsi la requête DQL créée avec cette méthode :

```
$qb = $this->_em->createQueryBuilder();
$qb->select('a')
   ->from('inscriptionVisiteurBundle:Accompagnateur', 'a')
   ->where('a.inscription = :id')
     ->setParameter('id', $inscription);
```

est équivalente à cette requête SQL :

```
SELECT *
FROM Accompagnateur a
WHERE a.inscription = [inscription];
```

Cependant le QueryBuilder présente de nombreuses limites à mon goût. Ainsi je n'ai trouvé aucun moyen de faire une requête comme ceci :

```
SELECT *
FROM Accompagnateur a
WHERE a.inscription = [inscription]
AND (a.raison = 'both' OR a.raison = [raison]);
```

Les méthodes `andWhere()` et `orWhere()` existent mais je n'ai trouvé aucun moyen de les hiérarchiser, ce qui rend impossible la création d'une requête telle que celle ci-dessus. Il est bien sûr possible de réaliser une telle requête mais seulement en injectant directement la requête dans le QueryBuilder, ce que je trouve regrettable car brisant la philosophie de Doctrine de base de données gérées uniquement par des objets.

4.6 Bilan du projet

Ce projet aura été à mon goût le plus intéressant de ce stage.

Il m'aura réconcilié avec le développement web, matière que je n'affectionnais pas énormément jusque là. Mais Symfony m'a étonnée par sa puissance et sa flexibilité.

J'ai trouvé ce framework peu aisé à prendre en main au départ, dû justement à sa taille et sa puissance, mais après avoir travaillé avec je n'imagine absolument plus développer une application web sans l'utiliser. Il permet d'avoir un code clair, de gérer une base de données de manière transparente et d'avoir des url compréhensibles pour les humains et non plus une suite de symboles sans signification aucune.

De même j'ai trouvé la gestion des utilisateurs étonnamment simple et intuitive.

Doctrine m'a permis de tester une nouvelle manière de manipuler les données, mais comme je l'ai expliqué dans la section précédente, ses limites dans Symfony m'ont déçues.

Conclusion

Cette première expérience d'informatique en milieu professionnel m'aura principalement permis de me rendre compte de la diversité de situations auxquelles nous pouvons être confrontés.

En effet j'ai eu la chance de travailler sur trois projets distincts et donc d'avoir des interlocuteurs différents ayant chacun leurs besoins et leurs objectifs.

Je regrette de ne pas avoir pu mener à bien le déploiement de PStage mais cela m'aura appris à quel point il est facile pour un projet de prendre du retard à cause d'un problème de communication entre services.

Bilan technique et personnel

J'ai eu à travailler tout au long de ce stage avec des outils que je ne connaissais ou ne maîtrisais pas forcément en arrivant, et je suis donc ravie d'avoir non seulement pu mettre en pratique des notions étudiées en cours, mais aussi d'avoir pu acquérir de nouvelles connaissances et de nouvelles compétences.

Ainsi j'ai pu améliorer mes compétences en PHP objet, élément du langage que nous n'avions pas eu la chance de voir en cours, mais aussi en JDBC.

J'ai appris à utiliser de nouveaux outils, tels que le moteur de servlet Tomcat que je ne connaissais absolument pas auparavant, les annuaires LDAP, la librairie FPDF ou encore le logiciel VSphere Client, permettant de gérer des machines virtuelles basées sur la technologie VMware.

L'élément le plus bénéfique à mon goût de ce stage est la possibilité que j'ai eue d'apprendre à utiliser un des frameworks les plus reconnus actuellement, à l'heure où le développement web sans utilisation de briques logicielles devient anecdotique et de moins en moins justifié. Ceci m'aura permis de me familiariser avec le patron de conception Modèle-Vue-Contrôleur, avec la gestion d'une base de données à travers un ORM ou encore à apprendre à utiliser un moteur de template tel que Twig.

J'estime que j'ai également eu beaucoup de chance de pouvoir différentes méthodes de gestion des données, j'ai ainsi pu comparer les avantages et inconvénients de chacune afin de pouvoir choisir lors d'un futur projet la méthode la plus adaptée à celui-ci. J'ai ainsi pu constater que Doctrine est un outil très puissant mais que je ne conseillerai que modérément en cas de requêtes complexes à exécuter, tout comme un annuaire LDAP qui est selon moi adapté pour un volume faible de données.

Enfin sur le plan personnel, malgré le fait que je n'ai eu aucune difficulté à m'intégrer sur mon lieu de travail, ce stage m'aura confortée dans l'idée de continuer mes études. En effet, outre le fait que je souhaite me spécialiser dans la conception et la gestion de base de données et donc acquérir de nouvelles connaissances, la vie étudiante m'attire encore bien plus que la vie de bureau.