Jallal Boudabza
Yacine Maghezzi

Master Informatique 2eme année

Département de Mathématique et Informatique

Systèmes d'Informations Orientés Objets
Rapport Final de Projet

Délai de remise : 16 Janvier 2014

Encadrant : Kokou Yetongnon

Table des matières

1 Introduction

De nos jours, avec l'explosion des systèmes connectés et la numérisation de l'information, la base de données est un mal nécessaire afin de pouvoir ranger de manière structurée les informations. Dans ce rapport, notre ambition est de pouvoir expliquer nos choix, nécessaire à la réussite du projet. Pour rappel, nous devons créer un système d'informations orienté objet pour la gestion de *l'université libre de Quetigny*, fictive bien évidemment. Il est a noter que les choix que nous expliquerons tout au long de ce rapport, ne sont pas définitifs, mais seront suivis dans la grande majorité.

2 Pré-Requis

FIGURE 2.1 – Logo de notre application

Dans ce projet, nous allons implémenter un système d'informations complet, qui comprendra plusieurs modules de développement. L'idée est de modéliser un système complet, pouvant être déployé sur une structure donnée. L'application doit être fonctionnelle et permettre de gérer cette université dont nous avons crée le logo ci dessus. Nous accorderons une grande importance au choix de l'implémentation du système en essayant de se rapprocher le plus possible d'une application délivrée par par des professionnels. Le choix de la gestion de la persistance des données sera expliqué dans les parties ci dessous.

2.1 Langage de programmation

Le langage que nous utiliserons est java. En effet ce langage nous permettra une connexion à différentes bases de données de manière simplifiée. La richesse de la documentation, et les interfaces entre les différentes SGBD nous permettent d'obtenir une certaine flexibilité. Java nous fourni un panel de drivers d'interfacage entre l'application Web, et le SGBD choisi. On citera par exemple *JDBC*, utile pour les bases de données traditionnelles ou encore Oracle, mais bien d'autres encore qui nous permettront d'interroger une éventuelle base NoSQL.

3 Organisation et Spécifications

3.1 Gestion du temps

Afin de pouvoir montrer de manière simple et explicite la gestion du temps de ce projet, nous avons effectué un diagramme dit de "Gantt", regroupant les dates et étapes approximatives qui ont conduit à la réalisation de celui ci.

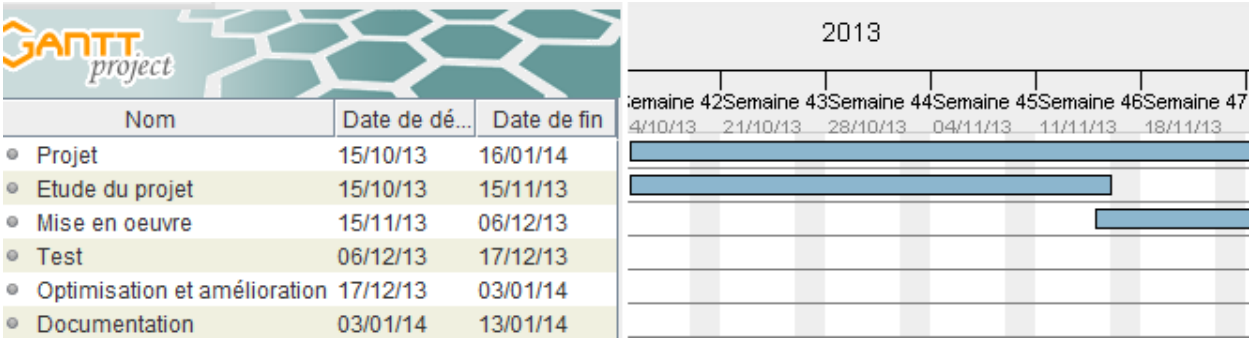


FIGURE 3.1 – Diagramme de Gantt Partie 1

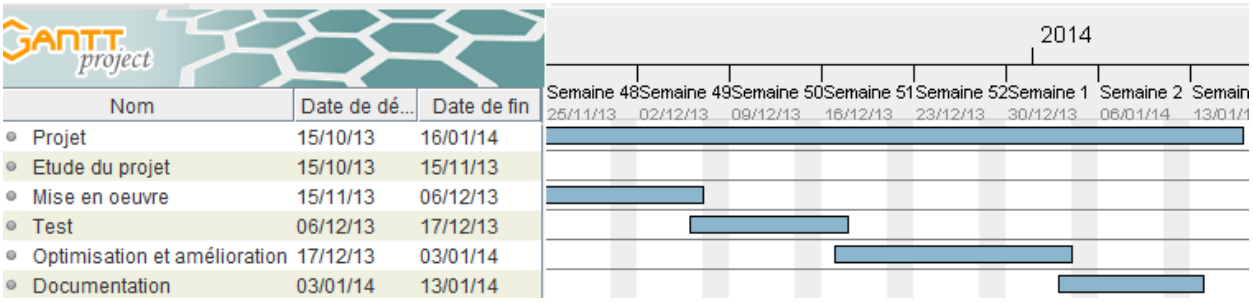


FIGURE 3.2 – Diagramme de Gantt Partie 2

3.2 Vue d'ensemble

Afin de que ce projet soit réalisé dans de bonnes conditions, l'utilisation de certains logiciels, langages et concepts a été nécessaire. En effet, ce projet a été majoritairement programmé en JAVA. L'utilisation de ce langage dit "orienté objet" nous a permis de structurer nos données, et s'y prêtait parfaitement dans les cas que nous devions traiter. De plus, l'exploitation du format de données renvoyés par nos requêtes sur mongoDB par JAVA est assez simple d'utilisation, donc argument de plus en faveur de JAVA.

Structurellement parlant, le client web développé est censé se situer sur un serveur situé lui même sur une machine virtuelle. En effet, cette spécification nous a forcé à créer une application qui, destinée au *smartphones*[1], se devait d'être légère et facile d'exécution, sans demander trop de ressources. La plupart des *smartphones* ne sont pas encore doté de processeurs double cœur avec une mémoire vive supérieur à 512MB.

C'est la raison pour laquelle nous devions créer une application qui serait exécutée sur le serveur, qui ferait les traitements, et renverrai l'HTML généré par les JSP. Le code serait donc invisible, et apparaitrait comme du code statique au yeux de l'utilisateur. Raison pour laquelle nous utilisons l'ensemble JSP/Servlet[2]. L'utilisation d'un IDE[3] tel que Netbeans nous as donc aidé à la réalisation de ce système, car celui ci possède des options tels que la correction dynamique d'erreur, la génération de fonction interrogeant le BDD, et l'auto-indentation[4](for utile pour la lisibilité du code). Ce projet m'a permis aussi de découvrir de nouveau outils, qui se sont avéré indispensable quand à la synchronisation du code avec les personnes ayant intervenu sur le projet. J'ai nommé GitHub. La suite de ce rapport vous apportera precision sur ce que sont ces outils, ainsi que leur utilisations dans la réalisation du projet.

3.3 Matériel et Logiciels

3.3.1 Netbeans

NetBeans est un environnement de développement intégré (IDE[5]), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2. En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web). Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC).

1. Un smartphone ou téléphone intelligent, est un téléphone mobile
2. Explication dans la suite du rapport
3. Un IDE est un programme regroupant un ensemble d'outils pour le développement de logiciels.
4. L'indentation est l'action qui permet d'ajouter des caractères de tabulation dans un fichier texte
5. Interface De Développement

3.3.2 Mozilla et cie

Mozilla Firefox est un navigateur Web libre et gratuit, développé et distribué par la Mozilla Foundation avec l'aide de centaines de bénévoles grâce aux méthodes de développement du logiciel libre/open source[6] et à la liberté du code source.

Les nombreux plugins[7] ainsi que les outils d'analyse que propose ce navigateur m'ont permis de debbuger mon code. Plus particulièrement Firebug, qui m'a permit d'analyser les entêtes de mes requêtes à mon Servlet, ainsi que les temps de réponses de celui.

3.3.3 Github

GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git.

Le site dit de "*Social Coding*[8]" permet a tout développeur d'avoir une aide tout au long du développement de son projet, grâce à la gestion de version que celui ci propose. Cet outil suit l'évolution des fichiers source et garde les anciennes versions de chacun d'eux.

L'utilisation de cet outil m'a permit d'avoir un aperçu de l'avancement de notre projet, ainsi que l'affichage de plusieurs statistiques en tout genre. Cela nous a également permit d'avoir une copie de mon travail sur un serveur distant en cas de crash de nos machines.

Il existe plusieurs types de logiciels proposant le même type de service comme SVN, Mercurial, CVS ... Mais Git nous a semblé le plus approprié et le plus facile a prendre en main.

Ces logiciels sont fortement conseillés pour gérer un projet informatique.

3.3.4 Tomcat

Apache Tomcat est un conteneur web libre de servlets et JSP Java EE. Il implémente les spécifications des servlets et des JSP du Java Community Process, est paramétrable par des fichiers XML et de propriétés, et inclut des outils pour la configuration et la gestion. Il comporte également un serveur HTTP. De plus il va de pair avec NetBeans 7.4 ce qui nous facilite le déploiement.

3.4 Les langages

3.4.1 JAVA

Le langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems. Ses caractéristiques ainsi que la richesse de son écosystème et de sa communauté lui ont permis d'être très largement utilisé pour le développement d'applications de types très disparates. Java est notamment largement utilisé pour le développement d'applications d'entreprise et mobiles.

6. La désignation open source s'applique aux logiciels dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire la possibilité de libre redistribution, d'accès au code source et aux travaux dérivés.

7. Un plugin, aussi nommé module d'extension, est un paquet qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités.

8. Code source libre service

3.4.2 JavaServer Pages

Le JavaServer Pages ou JSP est une technique basée sur Java qui permet aux développeurs de créer dynamiquement du code HTML, XML ou tout autre type de page web. Cette technique permet au code Java et à certaines actions prédéfinies d'être ajoutés dans un contenu statique. Depuis la version 2.0 des spécifications, la syntaxe JSP est complètement conforme au standard XML. En clair, un fichier JSP génère du code HTML, en effectuant les actions définies par le code JAVA qui est contenu dans les balises notées "<En récupérant les informations que le Servlet [9] me renvoi, nous effectuons les traitements sur les objets récupérés. En regardant le code source de la page, elle apparait comme une page HTMl statique.

Fonctionnement des JSP

Contrairement à son homologue PHP, lorsqu'une page JSP a été demandée, elle reste en mémoire pour que les autres demandes soient traitées plus rapidement. L'interprétation d'une page contenant du code JSP se fait de la manière suivante :

- L'utilisateur demande la page JSP via son client web qui exécute une requête HTTP.
- Le serveur web comprend la requête et transféré la demande au conteneur de servlets.
- Le conteneur de servlet vérifie si la page JSP est déjà en mémoire. Si ce n'est pas le cas, il génère le code Java puis le compile. Sinon, le conteneur de servlet exécute directement la classe Java correspondant à la page JSP.
- Ensuite le code HTML résultant est transféré au serveur web.
- Enfin, le serveur renvoi le code HTML au client.

Le schéma suivant illustre ces différentes étapes :

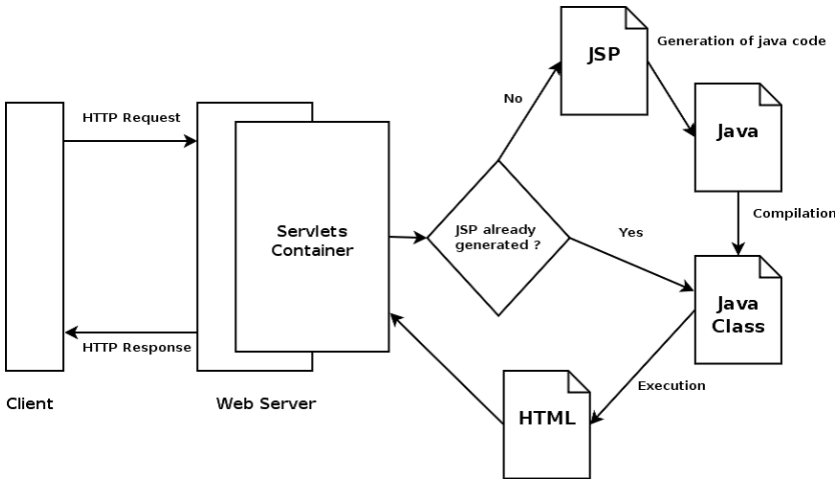


FIGURE 3.3 – Demande d'une page JSP

3.5 Frameworks et Libraires

3.5.1 jQuery Mobile

jQuery Mobile est un *framework* [10] web optimisée pour Smart-phones actuellement développé par l'équipe du projet jQuery. Le développement se concentre

9. Voir la partie
10. un framework est un kit de composants logiciels

sur la création d'un *framework* compatible avec une grande variété de smart-phones et tablettes, chose rendue indispensable a cause de l'explosion des marché des plateformes mobiles. Grâce à ce *framework*, nous avons pu créer la partie client de **ULQ**, de manière simple intuitive, et optimisée pour les plateformes mobiles [11].

3.6 Concepts utilisés

3.6.1 Les Servlets

Un servlet est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Ces données sont le plus généralement présentées au format HTML, mais elles peuvent également l'être au format XML ou tout autre format destiné aux navigateurs web. Les servlets utilisent l'API Java Servlet (package javax.servlet).

Un servlet s'exécute dynamiquement sur le serveur web et permet l'extension des fonctions de ce dernier, typiquement : accès à des bases de données, transactions d'e-commerce, etc. Un servlet peut être chargée automatiquement lors du démarrage du serveur web ou lors de la première requête du client. Une fois chargées, les servlets restent actives dans l'attente d'autres requêtes du client.

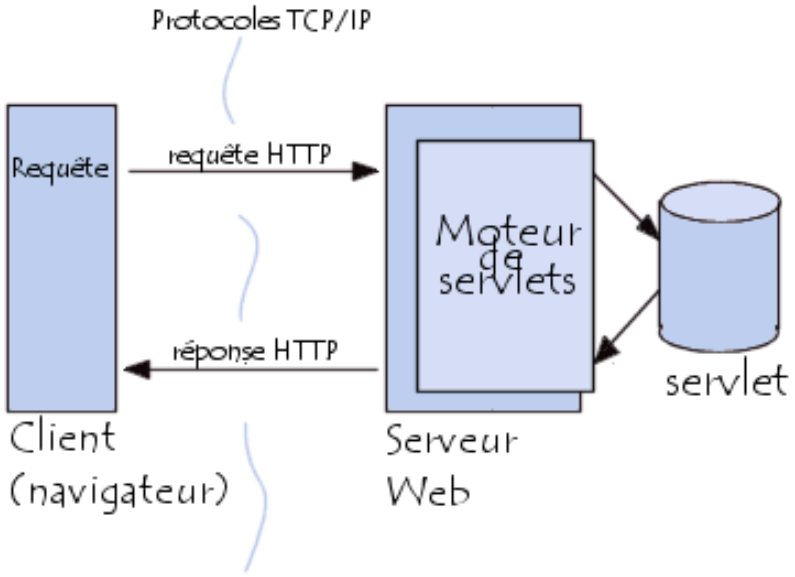


FIGURE 3.4 – Fonctionnement d'un servlet

11. tels que Android, Bada, iOS, Symbian ...

3.6.2 Base de donnée NoSQL

En informatique, NoSQL (Not only SQL en anglais) désigne une catégorie de systèmes de gestion de base de données (SGBD) qui n'est plus fondée sur l'architecture classique des bases relationnelles. L'unité logique n'y est plus la table, et les données ne sont en général pas manipulées avec SQL. À l'origine, servant à manipuler des bases de données géantes pour des sites web de très grande audience tels que Google, Amazon.com, Facebook ou eBay1, le NoSQL s'est aussi étendu par le bas après 2010. Il renonce aux fonctionnalités classiques des SGBD relationnels au profit de la simplicité. Les performances restent bonnes avec la montée en charge (scalabilité) en multipliant simplement le nombre de serveurs, solution raisonnable avec la baisse des coûts, en particulier si les revenus croissent en même temps que l'activité.

4 Base de données

4.1 Choix du SGBD

MongoDB est un système de gestion de base de données orientée documents, répartissable sur un nombre quelconque d'ordinateurs, efficace pour les requêtes simples, et ne nécessitant pas de schéma prédéfini des données. **MongoDB** fait parti des SGBD dit NoSQL. Cependant le vrai intérêt de MongoDB par rapport aux concurrents, c'est sa gestion de requêtages. On peut requêter de façon très fine grâce à une grande quantité de mots-clé. On peux donc faire des requêtes aussi riches qu'en SQL, mais tout ça sur une base de données orientée documents.

4.2 Diagramme de classes

Voici le diagramme de classe général de l'application. Afin de mieux comprendre le fonctionnement ainsi que l'architecture de notre base de données, il est nécessaire de faire un diagramme de classes. Les relations, classes ainsi que les cardinalités seront en grande partie respectés sauf cas de force majeur.

Les figures 4.1 et 4.2 correspondent donc au diagramme de classes. Le diagramme a été scindé en 2 parties plus ou moins logiques, dont nous allons faire l'explication.

Dans cette première partie du diagramme de classes (Figure 4.1), nous voyons les classes suivantes :

- Personne (est hérité par Personnel et par Etudiant et est associé à Cours)
- Etudiant (hérite de la classe Personne)
- Bureau (est associée à Personnel)
- LabTP (est associée à Bureau)
- Campus (est associé à Personne)
- Personnel (hérite de Personne et est hérité par Administrateur, Technicien, ChargeCours et Tuteur)
- Administrateur (hérite de Personnel)
- Technicien (hérite de Personnel)
- Tuteur (hérite de Personnel)
- ChargeCours (hérite de Personnel, est héritée par EnseignantAssocie et Enseignant- Vacataire et est associée à Cours)
- EnseignantAssocie (hérite de ChargeCours)

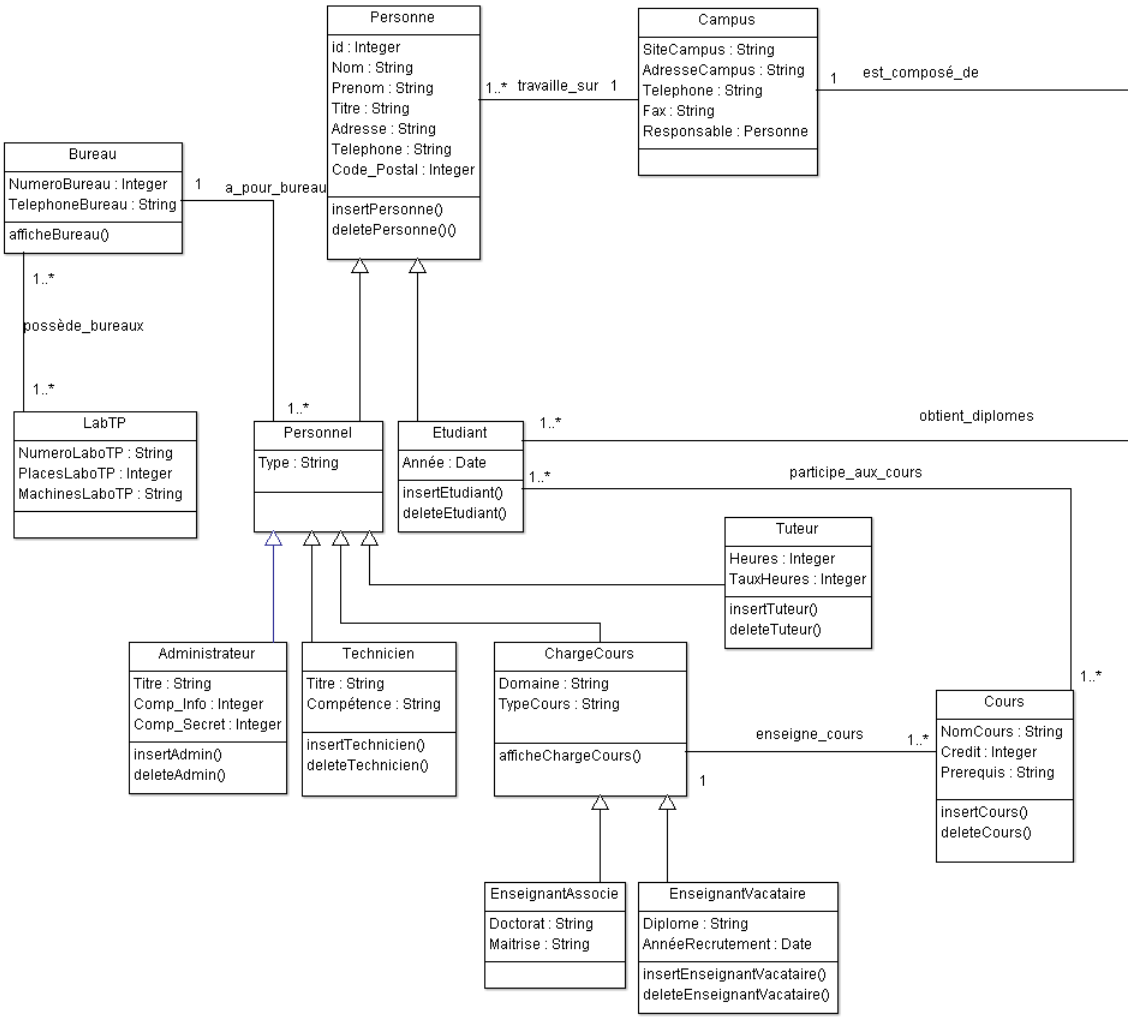


FIGURE 4.1 – Partie gauche du diagramme

– EnseignantVacataire (hérite de ChargeCours)
– Cours (est associé à ChargeCours)

Dans la figure 4.1 nous remarquons qu'il y a 6 classes provenant d'héritage. En effet, Administrateur, Technicien, ChargeCours, Tuteur héritent toutes les 4 de Personnel. Il y a aussi EnseignantVacataire et EnseignantAssocie qui hérite de la classe ChargeCours. Donc le personnel est composé d'administrateurs, de techniciens, de chargés de cours et de tuteurs. Un chargé de cours peut être un enseignant vacataire ou un enseignant associé. Un chargé de cours (enseignant vacataire ou associé) enseigne plusieurs cours, un cours est enseigné par un chargé de cours. Un étudiant participe à des cours, un cours est suivi par plusieurs étudiants. Un étudiant obtient des diplômes, un diplôme peut être obtenu par plusieurs étudiants.

Il existe une association entre Personnel et Bureau car un membre du personnel a un bureau, et un bureau peut appartenir à un ou plusieurs membre du personnel. Une personne fréquente un campus, mais un campus peut être fréquenté par plusieurs personnes.

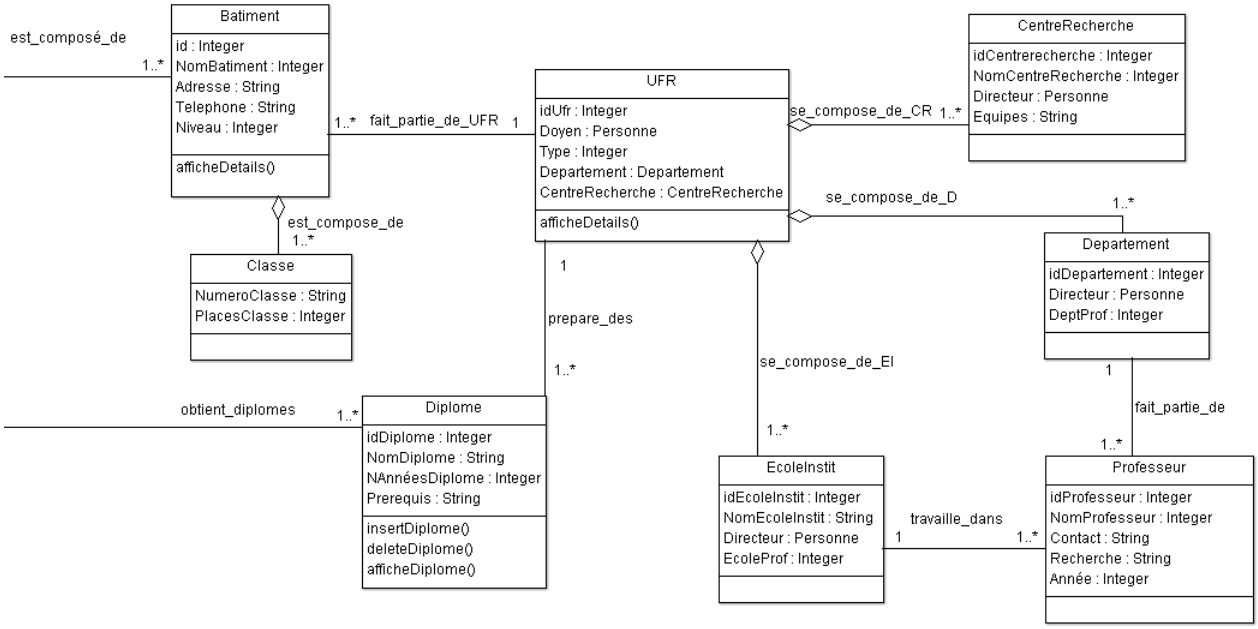


FIGURE 4.2 – Partie droite du diagramme

Dans cette deuxième partie du diagramme de classes (Figure 4.2), nous voyons les classes suivantes :

- Batiment (est composé de Classe)
- Classe (compose un Batiment)
- UFR (est associée à Diplome, Batiment et est composé de EcoleInstitut, CentreRecherche et Departement)
- Diplome (est associé à UFR et Etudiant)
- CentreRecherche (compose l'UFR)
- Departement (compose l'UFR et est associé à Professeur)
- EcoleInstitut (compose l'UFR et est associé à Professeur)
- Professeur (est associé à Departement et à EcoleInstitut)

Un campus est composé de plusieurs bâtiments, un bâtiment faire partie d'un campus. Les bâtiment d'un campus font partie d'une UFR. Un bâtiment est composé de plusieurs classes.

Une UFR permet de préparer des diplômes, un diplôme est obtenu dans une UFR en particulier. Une UFR se compose de centres de recherche, de département et d'école institution. Et un professeur travaille dans une école institution et fait partie d'un département. Un département et une école institution ont chacun plusieurs professeurs.

4.3 Modele ODMG

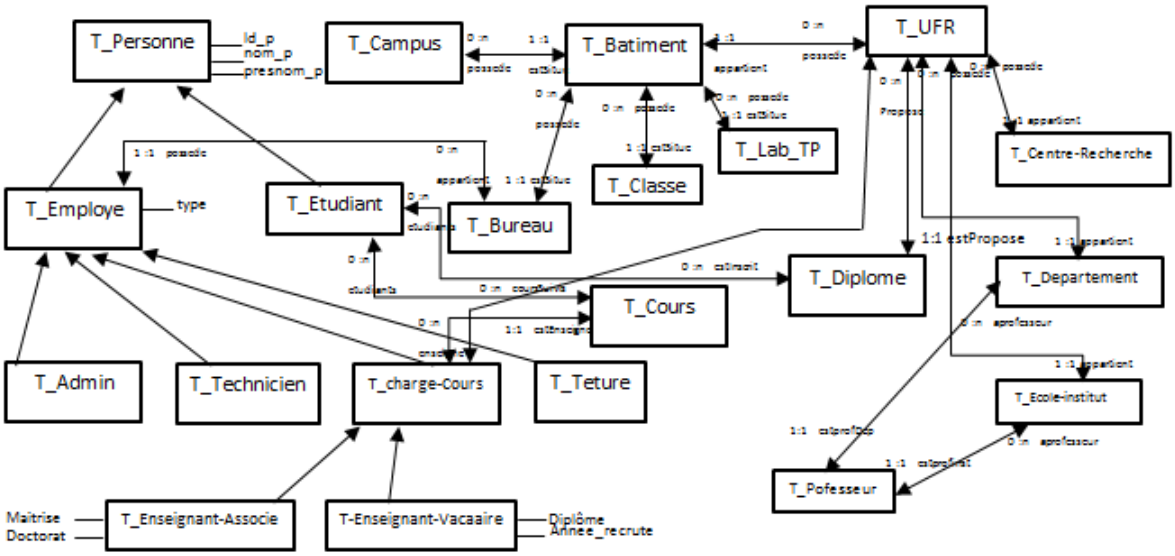


FIGURE 4.3 – Diagramme ODMG

4.4 La syntaxe ODL

Afin de mettre en œuvre la conception de notre base de donnes orientées objets *U-Quetigny* nous allons décrire les différentes classes utilisées.

Tout d'abord, nous avons la classe *Personne* :

```
class Personne
    (extent LesPersonnes key idPersonne)
{
    attribute int id;
    attribute string Nom;
    attribute string Prenom;
    attribute string Titre;
    attribute string Adresse;
    attribute string Telephone;
    attribute string Code_Postal;

    relationship List<Campus> travaille_sur inverse Campus::a_pour_professeur;

    void insertP(idPersonne);
    void deleteP(idPersonne);
}
```

Dans la classe *Personne* nous retrouvons la déclaration des différents attributs tels que *idPersonne*, *Nom-Personne* ou encore *PrenomPersonne*. Tous ces attributs sont décrits dans le diagramme de classes de la figure **??**.

Nous retrouvons également les méthodes suivantes : *insertPersonne* et *deletePersonne*, chacune de ses méthode prend en paramètres l'identifiant de la personne et l'insère ou la supprime de la base de donnes. Nous avons aussi la déclaration des liens entre les classes, par exemple la classe *Personne* est associée à la classe *Campus*, en effet nous pouvons affirmer qu' *une personne travaille sur un campus* et qu'*un campus a pour professeurs*

Ensuite, nous nous intéresserons aux classes *Personnel* et *Etudiant* du diagramme de la figure 4.1 car elles héritent de la classe *Personne*. Or il est intéressant et nécessaire de savoir gérer ce genre de classes.

Classe *Personnel* :

```
class Personnel : Personne
{
    attribute string TypePersonnel;

    relationship List<Bureau> a_pour_bureau inverse Bureau::appartien_a;
}
```

Classe *Etudiant* :

```
class Etudiant : Personne
{
    attribute date Annee;

    relationship Set<Diplome> obtient_diplomes inverse Diplome::est_obtenu_par;
    relationship Set<Cours> suit_cours inverse Cours::est_suivi_par;

    void insertEtudiant();
    void deleteEtudiant();
}
```

La classe *Personnel* possède également des classes filles.
Classe *Administrateur* :

```
class Administrateur : Personnel
{
    attribute string Titre;
    attribute int Comp_Info;
    attribute int comp_secret;

    void insertAdmin();
    void deleteAdmin();
}
```

Classe *Technicien* :

```
class Technicien : Personnel
{
    attribute string Titre;
    attribute string Competence;

    void insertTechnicien();
    void deleteTechnicien();
}
```

Classe *Tuteur* :

```
class Tuteur : Personnel
{
    attribute int Heures;
    attribute int TauxHeures;

    void insertTuteur();
    void deleteTuteur();
}
```

Classe *ChargeCours* :

```
class ChargeCours : Personnel
{
    attribute string Domaine;
    attribute string TypeCours;

    relationship Cours enseigne_cours inverse Cours::est_enseigne_par;

    void afficheChargeCours();
}
```

La classe *ChargeCours* possède également des classes filles.

Classe *EnseignantAssocie* :

```
class EnseignantAssocie : ChargeCours
{
    attribute string Doctorat;
    attribute string Maitrise;
}
```

Classe *EnseignantVacataire* :

```
class EnseignantVacataire : ChargeCours
{
    attribute string Diplome;
    attribute date AnneeRecrutement;

    void insertEnseignantVacataire();
    void deleteEnseignantVacataire();
}
```

Classe *Campus* :

```
class Campus
{
    attribute string SiteCampus;
    attribute string AdresseCampus;
    attribute string Telephone;
    attribute string Fax;
    attribute Personne Responsable;

    relationship Personne est_frequente_par inverse Personne::frequente;
    relationship Batiment est_compose_de inverse Batiment::fait_partie_du_campus;
}
```

Classe *Cours* :

```
class Cours
    (extent LesCours key idCours)
{
    attribute int idCours;
    attribute string NomCours;
    attribute int Credit;
    attribute string Prerequis;

    relationship List<ChargeCours> est_enseigne_par inverse ChargeCours::enseigne_cours;
    relationship Set<Etudiant> est_suivi_par inverse Etudiant::suit_cours;

    void insertCours();
    void deleteCours();
}
```

Classe *Diplome* :

```
class Diplome
    (extent LesDiplomes key idDiplome)
{
    attribute int idDiplome;
    attribute string NomDiplome:
    attribute int NAnneesDiplome;
    attribute string Prerequis;

    relationship Set<Etudiant> est_obtenu_par inverse Etudiant::obtient_diplomes;
    relationship List<UFR> est_prepare_dans inverse UFR::prepare;

    void insertDiplome();
    void deleteDiplome();
    void afficheDiplome();
}
```

Classe *UFR* :

```
class UFR
    (extent LesUFR key idUFR)
{
    attribute int idUFR;
    attribute Personne Doyen;
    attribute int TypeUFR;
    attribute Departement DepartementUFR;
    attribute CentreRecherche CentreRechercheUFR;

    relationship Batiment est_compose_de_batiments inverse Batiment::fait_partie_de_UFR;
    relationship Diplome prepare_diplomes inverse Diplome::est_prepare_dans;

    relationship EcoleInstit se_compose_decole inverse EcoleInstit::ecole_fait_partie_de;
    relationship CentreRecherche se_compose_de_centre inverse CentreRecherche::
        centre_fait_partie_de;
    relationship Departement se_compose_de_dep inverse Departement::dep_fait_partie_de;

}
```

Classe *CentreRecherche* :

```
class CentreRecherche
    (extent LesCentresRecherche key idCentreRecherche)
{
    attribute int idCentreRecherche;
    attribute string NomCentreRecherche;
    attribute Personne DirecteurCentreRecherche;
    attribute string Equipes;

    relationship List<UFR> centre_fait_partie_de inverse UFR::se_compose_de_centre;
}
```

Classe *Departement* :

```
class Departement
    (extent LesDepartement key idDepartement)
{
    attribute int idDepartement,
    attribute Personne DirecteurDepartement;
    attribute int DeptProf;

    relationship List<UFR> dep_fait_partie_de inverse UFR::se_compose_de_dep;
    relationship Professeur dep_a_pour_professeur inverse Professeur::fait_partie_de
}
```

Classe *EcoleInstit* :

```
class EcoleInstit
    (extent LesEcolesInstit key idEcoleInstit)
{
    attribute int idEcoleInstit;
    attribute string NomEcoleInstit;
    attribute Personne DirecteurEcoleInstit;
    attribute int EcoleProf;

    relationship List<UFR> ecole_fait_partie_de inverse UFR::se_compose_decole;
    relationship Professeur ecole_a_pour_professeur inverse Professeur::travaille_dans
}
```

Classe *Professeur* :

```
class Professeur
    (extent LesProfesseurs key idProfesseur)
{
    attribute int idProfesseur;
    attribute string NomProfesseur;
    attribute string Contact;
    attribute string Recherche;
    attribute int Annee;

    relationship List<EcoleInstit> travaille_dans inverse EcoleInstit::
        ecole_a_pour_professeur;
    relationship List<Departement> fait_partie_de_dep inverse Departement::
        dep_a_pour_professeur;
}
```

4.5 Insertion de tuples

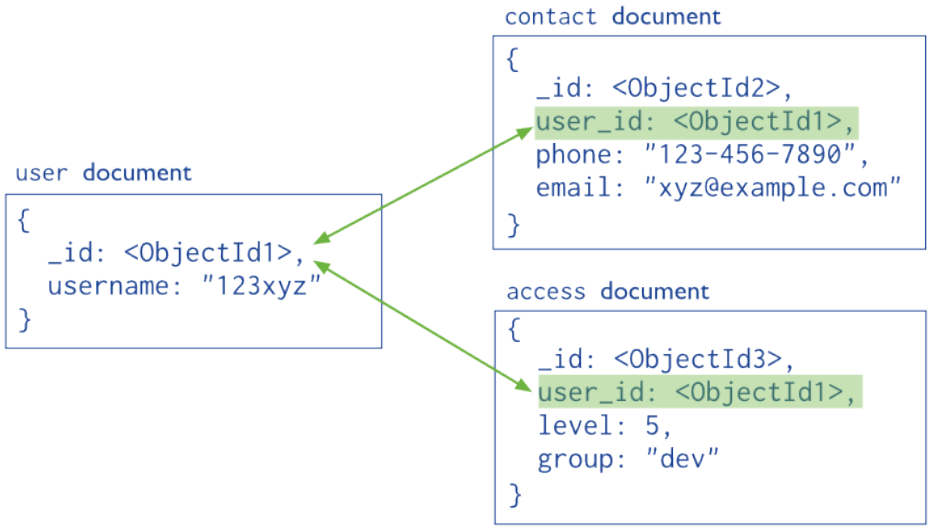


FIGURE 4.4 – Fonctionnement de MongoDB

MongoDB est un SGBD orienté document. Il est donc important d'en comprendre le fonctionnement. Les données sont liées entre elle via des références entre entités, comme expliques dans le schémas précédent.

Voici un exemple d'insertion dans la classe Personne sur **MongoDB** :

```
db.personne.insert( { nom: "Boudabza", prenom: "Jallal", Titre: "Monsieur", Adresse: "Chemin
    de st Apo" ....}
```

4.6 Requêtes OQL

Voici l'exemple de quelques requêtes sur le schémas crée précédemment.

Requête n°1

```
SELECT struct (p.nom, p.prenom)
FROM p in Personne
GROUP BY (Dijonnais: p.codepostal = 21000 ,
Autre : p.codepostal != 21000);
```

Explication : Affichage des de toutes les personnes provenant de Dijon et des alentours.

Requête n°2

```
SELECT c.* FROM etudiant e, cours c WHERE e.id = c.idCours
 AND e.nom = "Maghezzi" AND e.prenom = "Yacine";
```

Explication : Affichage des cours de l'étudiant Maghezzi Yacine.

La requête sur **MongoDB** donnera :

```
etu_temp = db.etu.findOne({nom : « Maghezzi », prenom : « Yacine »})
cours = db.cours.find({idCours : etu_temp.idCours})
```

4.7 Mise en place de MongoDB

La mise en place de la base mongoDB sous windows se fait tel un logiciel. L'archive de 200Mo se doit d'être décompressée dans un dossier. Pour ce faire, les deux commandes ci dessous sont nécessaires au démarrage du serveur MongoDB

```
D:\mongodb\bin\mongod.exe --dbpath d:\mongodb\data

D:\mongodb\bin\mongo.exe
```

MongoDb est maintenant prêt a recevoir des connections, comme dans la deuxième console, ou l'on se connecte directement au shell pour y introduire.

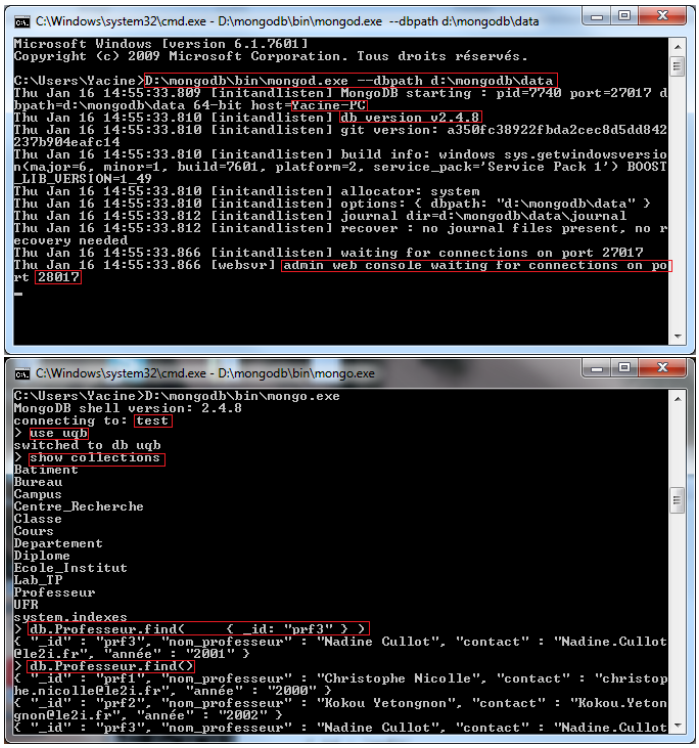


FIGURE 4.5 – Mise en place de mongoDb

4.8 Remplissage de la base MongoDB

```
db.Personne.insert(
  {_id : '1',
  type : 'Chargecours',
  nompersonne : 'Cullot',
  prenompersonne : 'Nadine',
  titrepersonne : 'Professeur',
  adressepersonne : 'Dijon',
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  domaine : 'informatique',
  typecours : 'basededonnees',
  campusid : 'c1'
  })

db.Personne.insert({_id : '21',
  type : 'Etudiant',
  nompersonne : 'Salat',
  prenompersonne : 'Aurélie',
  titrepersonne : 'Etudiant',
  adressepersonne : 'Dijon',
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  annee : '2012',
  campusid : 'c1'
  })

db.Personne.insert({_id : '22',
  type : 'Etudiant',
  nompersonne : 'Prautois',
  prenompersonne : 'Mathieu',
  titrepersonne : 'Etudiant',
  adressepersonne : 'Dijon',
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  annee : '2008',
  campusid : 'c1'
  })

db.Personne.insert({_id : '23',
  type : 'Etudiant',
  nompersonne : 'Simonot',
  prenompersonne : 'Jordan',
  titrepersonne : 'Etudiant',
  adressepersonne : 'Dijon',
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  annee : '2009',
  campusid : 'c1'
  })

  db.Personne.insert({_id : '31',
  type : 'Administrateur',
  nompersonne : 'adminnom',
  prenompersonne : 'adminprenom',
  titrepersonne : 'Administrateur',
  adressepersonne : 'Dijon',
```

```
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  titre : 'administrateur reseau',
  comptinfo : '5',
  compt-secret :'6',
  campusid : 'c1'
  })

  db.Personne.insert({_id : '41',
  type : 'Technicien',
  nompersonne : 'technom',
  prenompersonne : 'techprenom',
  titrepersonne : 'Technicien',
  adressepersonne : 'Dijon',
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  titre : 'Technicien 1',
  competence : 'competence technicien',
  campusid : 'c1'
  },

  db.Personne.insert({_id : '51',
  type : 'Tuteur',
  nompersonne : 'tuteurnom',
  prenompersonne : 'tuteurprenom',
  titrepersonne : 'Tuteur',
  adressepersonne : 'Dijon',
  telephonepersonne : '0381800000',
  codepostalpersonne : '21000',
  heures : 24,
  tauxheure : 0.8,
  campusid : 'c1'
  }
)

db.Campus.insert(
  {_id : 'c1',
  sitecampus : 'Site campus UB',
  adressecampus : 'Campus de Dijon ',
  telephone : '03812212121',
  fax : '0381212122',
  responsable : 'Responsable Campus UB'
  }
)

db.Batiment.insert(
  {_id : 'bat1',
  nombatiment : 'Mirande',
  adresse : 'Dijon',
  telephone : '0381111111',
  niveau : '4',
  campusid : 'c1',
  ufrid : 'ufr1'
  })

  db.Batiment.insert({_id : 'bat2',
  nombatiment : 'Gabriel',
  adresse : 'Dijon',
  telephone : '0381111111',
```

```
  niveau : '4',
  campusid : 'c1',
  ufrid : 'ufr2',
  ufrid : 'ufr3'
  }
)

db.UFR.insert(
  {_id : 'ufr1',
  nomUFR : 'UFR Sciences et Techniques',
  doyenUFR : 'Thierry Grison',
  typeUFR : 'Sciences',
  departementUFR : 'Scientifique',
  centrerechercheUFR : 'Sciences',
  batimentid :'bat1'
  })

  db.UFR.insert({_id : 'ufr2',
  nomUFR : 'UFR Droit et Sciences politiques',
  doyenUFR : 'Thierry Grison',
  typeUFR : 'Droit',
  departementUFR : 'Droit',
  centrerechercheUFR : 'Sciences politiques',
  batimentid :'bat2'
  })

  db.UFR.insert({_id : 'ufr3',
  nomUFR : 'UFR Sciences humaines',
  doyenUFR : 'Thierry Grison',
  typeUFR : 'Sciences humaines',
  departementUFR : 'Sciences humaines',
  centrerechercheUFR : 'Sciences humaines',
  batimentid :'bat2'
  }
)

db.Bureau.insert(
  {
  numerobureau : 'G218',
  telephonebureau : '0381000218',
  batimentid : 'bat1'
  })
  {
  numerobureau : 'GS17',
  telephonebureau : '0381000218',
  batimentid : 'bat1'
  })
  {
  numerobureau : 'GS16',
  telephonebureau : '0381000218',
  batimentid : 'bat1'
  })
  {
  numerobureau : 'GS15',
  telephonebureau : '0381000218',
  batimentid : 'bat1'
  }
)
```

```
db.Classe.insert(
  {
  numeroclasse : 'B400',
  placesclasse : '40',
  batimentid : 'bat1'
  })
  {
  numeroclasse : 'B300',
  placesclasse : '40',
  batimentid : 'bat1'
  })
  {
  numeroclasse : 'A200',
  placesclasse : '40',
  batimentid : 'bat1'
  }
)

db.LabTP.insert(
  {_id :'lab1',
  numerolab : 'A103',
  placestp : '37',
  machinestp : '35',
  batimentid : 'bat1'
  })
  db.LabTP.insert({_id :'lab2',
  numerolab : 'A104',
  placestp : '37',
  machinestp : '35',
  batimentid : 'bat1'
  })
  db.LabTP.insert({_id :'lab3',
  numerolab : 'A105',
  placestp : '37',
  machinestp : '35',
  batimentid : 'bat1'
  }
)

db.CentreRecherche.insert(
  {_id : 'cr1',
  nomcentrerecherche : 'Laboratoire Electronique, Informatique et Image',
  directeurcentrerecherche : 'Fabrice Meriaudeau',
  equipescentrerecherche : 'Checksem',
  UFRid : 'ufr1'
  }
)

db.Departement.insert(
  {_id : 'dep1',
  nomdepartement : 'departementinformatique',
  directeurdepartement : 'directeur departement informatique',
  profdepartement : 'professeur de departement',
  UFRid : 'ufr1'
  }
)

db.EcoleInstitut.insert(
  {_id : 'ei1',
```

```
  nomecoleinstitut : 'Ecole Superieur dIngenieurs de Recherche en Materiaux',
  directeurecoleinstitut : 'Pr. Gilles CABOCHE ',
  profecoleinstitut : 'professeur decole',
  UFRid : 'ufr1'
  })

  db.EcoleInstitut.insert({_id : 'ei2',
  nomecoleinstitut : 'Insitut de la Vigne et du Vin Jules Guyot',
  directeurecoleinstitut : 'Michèle GUILLOUX-BENATIER ',
  profecoleinstitut : 'professeur decole',
  UFRid : 'ufr1'
  })

  db.EcoleInstitut.insert({_id : 'ei3',
  nomecoleinstitut : 'Institut Universitaire de Technologie',
  directeurecoleinstitut : 'Jean-Michel LEFAURE',
  profecoleinstitut : 'professeur decole',
  UFRid : 'ufr1'
  }
)

db.Professeur.insert(
  {_id : 'prf1',
  nomprofesseur : 'Christophe Nicolle',
  contact : 'christophe.nicolle@le2i.fr',
  année : '2000'
  })

  db.Professeur.insert({_id : 'prf2',
  nomprofesseur : 'Kokou Yetongnon',
  contact : 'Kokou.Yetongnon@le2i.fr',
  année : '2002'
  })

  db.Professeur.insert({_id : 'prf3',
  nomprofesseur : 'Nadine Cullot',
  contact : 'Nadine.Cullot@le2i.fr',
  année : '2001'
  }
)

db.Cours.insert(
  {_id : 'cm1',
  nomcours : 'Outils de lIntelligence Artificielle',
  creditcours : '6',
  prerequiscours : 'L3'
  })

  db.Cours.insert({_id : 'cm2',
  nomcours : 'Systèmes Intelligents',
  creditcours : '6',
  prerequiscours : 'L3'
  })

  db.Cours.insert({_id : 'cm3',
  nomcours : 'Base de Donnees Multimedia',
  creditcours : '6',
  prerequiscours : 'L3'
  })
```

```
293
   db.Cours.insert({_id : 'cm4',
295  nomcours : 'Base de Donnes et Environnements Distribues',
   creditcours : '6',
297  prerequiscours : 'L3'
   })
299
   db.Cours.insert({_id : 'cm5',
301  nomcours : 'Systeme d'Informations Oriente Objet',
   creditcours : '6',
303  prerequiscours : 'L3'
   }
305 )

307 db.Diplome.insert(
   {_id : 'dip1',
309  nomdiplome : 'Maitrise STIC',
   nbanneesdiplome : '1',
311  prerequis : 'Licence'
   })
313
   db.Diplome.insert({_id : 'dip2',
315  nomdiplome : 'Licence STIC',
   nbanneesdiplome : '3',
317  prerequis : 'L2'
   })
319
   db.Diplome.insert({_id : 'dip3',
321  nomdiplome : 'Master Base de Donnees et Intelligence Artificielle',
   nbanneesdiplome : '1',
323  prerequis : 'M1'
   }
325 )
```

5 Mise en œuvre

5.1 Fonctionnement général

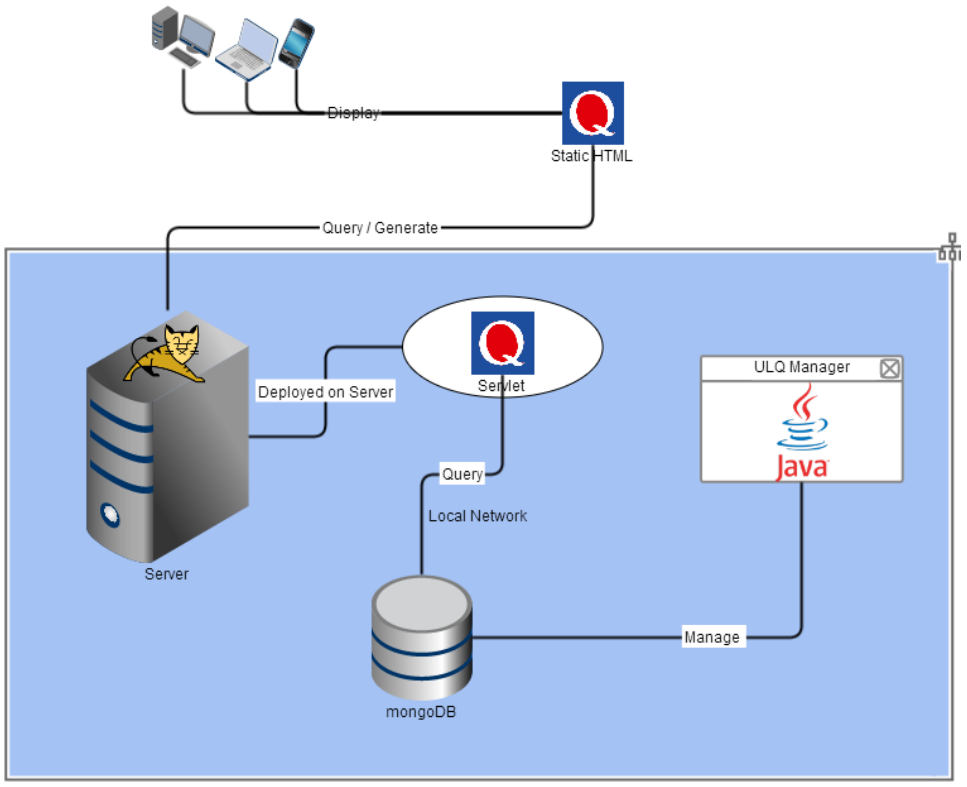


FIGURE 5.1 – Schémas Général

Afin de répondre aux besoins des utilisateurs, qu'il soient administrateurs ou autres, nous avons opté pour une application à deux vitesses.

En effet, par un soucis de sécurité et pour se préparer à des projets similaire dans une vie professionnelle nous avons préféré séparer l'application en deux parties. Une partie cliente dite "User-Friendly", agréable à voir pour les utilisateurs, qui aura un rôle de consultation. Grâce a cette application, l'utilisateur pourra seulement consulter la base de données NoSQL mise en place. Par soucis de professionnalisme, nous avons opté pour une interface optimisée pour SmartPhone et tablettes, avec des listes dynamiques. De plus notre application cliente permettra de explorer n'importe quelle autre base MongoDB existante. Elle est générique et pourra être réutilisée à l'avenir.

L'utilisation de servlets ici optimise les performances et la fluidité d'utilisation car tout le traitement se fait

en amont du client. Il ne verra qu'un code HTML statique. De plus nous avons appliqué a notre application une possibilité de recherche d'un tuple précis de la base.

Une seconde application dite administrateur, qui permettra aux personnes responsable de la base de donnée d'y faire des traitements plus pointu.

Il pourra bien évidemment consulter la BDD mais pourra surtout ajouter des tables/tuples et supprimer des tables/tuples.

L'application sera elle liée directement a la BDD. Elle est codée en Java.

5.2 Administrateur

5.2.1 Architecture des données

Pour cette application, nous avons décidé de faire une classe ConnectionMongoDB où toutes les fonctions seront répertoriées. De plus, nous avons réalisé plusieurs classes JFRAME pour l'interface.

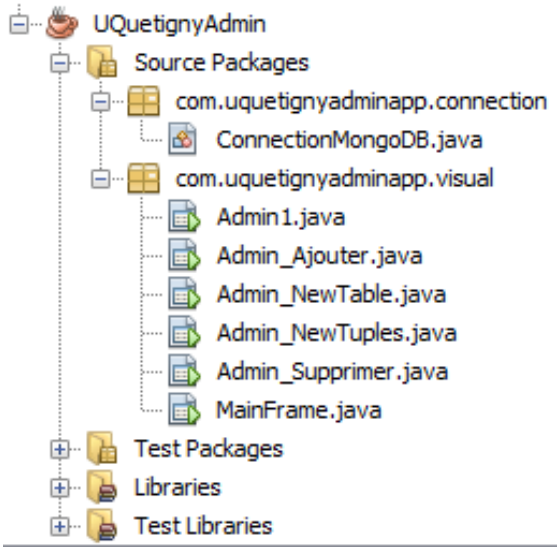


FIGURE 5.2 – Agencement des classes

5.2.2 Interface graphique

Pour l'interface graphique, nous avons créé plusieurs JFRAME, chaque JFRAME va permettre à l'utilisateur de faire une manipulation sur la base de données. En effet, le programme permet à l'administrateur de créer, supprimer des tables dans la base MongoDB. La classe ConnectionMongoDB .java permet de se connecter à la base de données MongoDB.

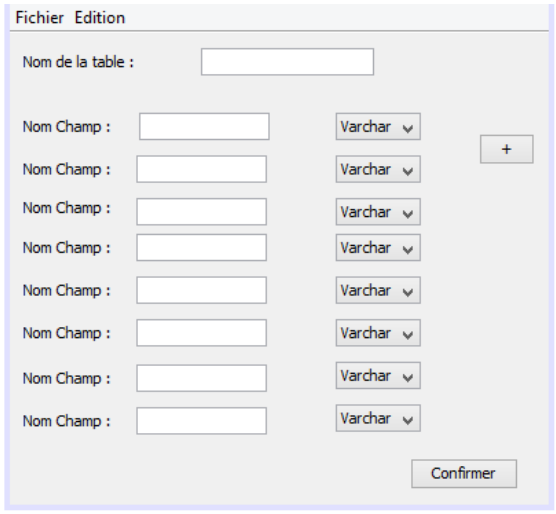


FIGURE 5.3 – Exemple

5.2.3 Connexion a la base de donnée

Dans la classe ConnectionMongoDB.java, nous avons développé quelques méthodes pour pour les réutiliser dans l'application. Pour se connecter à la base de données MongoDB, nous devons coder quelques lignes, en effet, après avoir créé notre base de données grâce au terminal, nous devons connecter MongoDB avec le projet Netbeans. Nous devons faire ceci :

```
public ConnectionMongoDB() throws UnknownHostException {
    Mongo mongoClient = new Mongo("localhost");
    db = mongoClient.getDB("test");
    //boolean auth = db.authenticate(myUserName, myPassword);

}

public DB getDB()
{
return db;
}
```

FIGURE 5.4 – Connexion a la base nommée "uqb"

Donc à chaque fois que nous allons devoir manipuler la base de données nous allons devoir faire appel à la classe ConnectionMondoDB() qui va se connecter a notre base de donnée MongoDB. Cette fonction est utilisée pour la suppression d"une table dans la classe AdminSupprimer.java

5.2.4 Méthodes utilisées

Methode de suppression d'une table

```
public void supprimerTable(String table) {
    DBCollection col = getRecordsOfSpecificCollection(table);
    DBCursor cursor =col.find();
    while (cursor.hasNext()) {
            col.remove(cursor.next());
    }
    col.drop();

}
```

FIGURE 5.5 – Connexion a la base nommée "uqb"

Pour cette méthode, nous récupérons un string que nous passons a un DBCollection, ensuite grâce a un DBCursor, MongoDB va pouvoir retrouver la table, ensuite nous avons juste besoin de faire un drop de cette collection mongoDB.

Méthode de récupération de toutes les tables de MongoDB

Nous avons aussi crée une méthode qui permet de récupérer fans une tableau toutes les tables de la base de donnée MongoDB. Cette méthode sera appelé plusieurs fois dans notre application.

```
public ArrayList<String> getAllClassOfDb() {
    ArrayList<String> list = new ArrayList<String>(db.getCollectionNames());
  //  System.out.println("Taille DB " +list.size());
    return list;
}
```

FIGURE 5.6 – Connexion a la base nommée "uqb"

5.3 Client

5.3.1 Architecture du projet

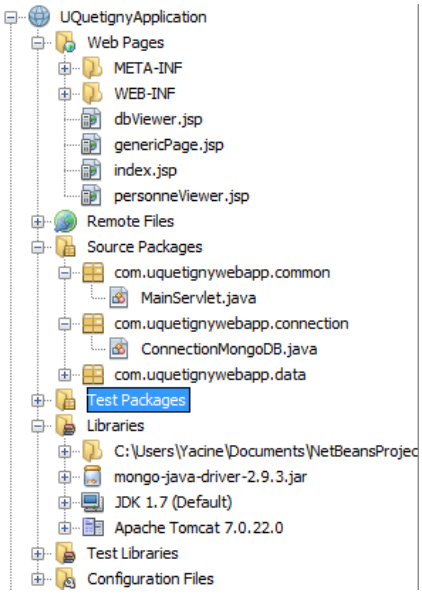


FIGURE 5.7 – Agencement des classes

Ici le projet est une application web, en effet, elle permettra de couter moins de ressources a la personne l'utilisant. Les pages JSP sont déstinées a l'affichage. Les différents packages common ,et connection sont respectivement utilisés pour interagir avec le client, et interagir avec la base de donnée.

5.3.2 Modele MVC

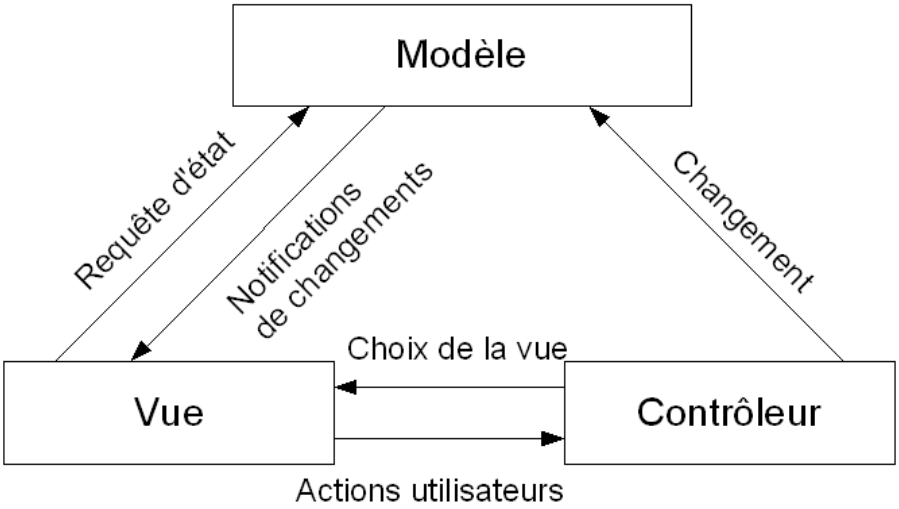
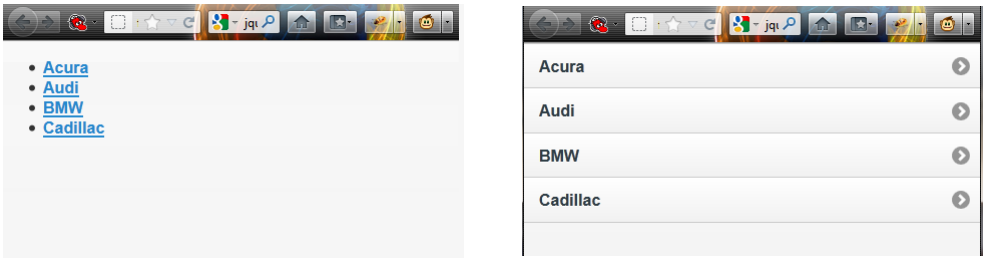


FIGURE 5.8 – Modele MVC

Malgré ce qui a été énoncé dans le rapport préliminaire le modèle MVC n'a pas été respecté. En effet le mapping des objets de la base mongoDb aurait rendu le projet trop rigide, et le passage d'objets complexes est bien difficile a mettre en oeuvre. Le controleur se matérialise bien par le Servlet, ainsi que la vue par les JSP cependant le modèle est exprimable via les arrayList que nous passons en paramètres à travers les pages.

5.3.3 Interface graphique

Le *framework* de jQuery est à la base de notre interface graphique. En effet, l'utilisation de ce *framework* optimisé pour les mobiles et appareils tactiles en tout genre, nous permet d'apporter une nouvelle dimension à notre application. Le développement du marché des *smartphones* nous contraint à prendre en compte cette variable. L'avantage c'est que ce *framework* s'adapte aussi aux navigateurs web sur nos machines de bureau tel que Firefox, Internet Explorer ... Le résultat est facile à percevoir d'autant que son utilisation est aisée. Il suffit d'ajouter certains attributs à des balises HTML pour obtenir un résultat plutôt esthétique.



(a) Sans jQuery (b) Avec jQuery

FIGURE 5.9 – Différence avec/sans jQuery mobile

5.4 Fichiers JSP

La fonctionnalité principale de cette application est d'afficher toutes les salles, ainsi que es informations les concernant. Pour ce faire, plusieurs pages JSP s'articule autour de cette fonctionnalité. La page d'accueil de mon application est généré par le fichier index.jsp. Cette page donne ensuite plusieurs choix possibles. La vue principale, qui sera accessible a tous, et par tous est décrite par le fichier dbViewer.jsp. Ce fichier affiche une liste non exhaustive de toutes les tables recensées dans la base de données de mongoDB. Une fois la table choisie, on passe comme paramètre le nom de salle au servlet qui redirige sur la page genericView.jsp nous affichant les information relative à la table en question.

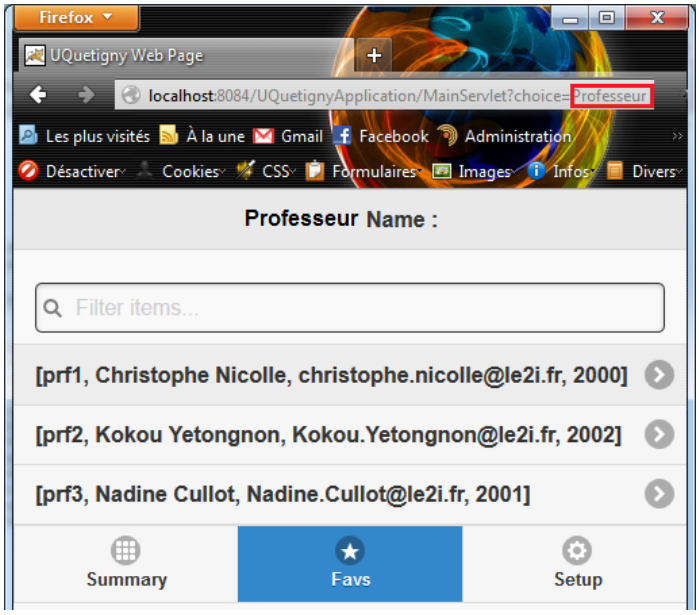


FIGURE 5.10 – Exemple pour la récupération des professeurs

5.5 Communication entre JSP et le Servlet

Le fonctionnement même de cette application repose sur le principe de communication entre le servlet, moteur de cette application, et les pages JSP. La communication se fait dans les deux sens.

- JSP vers Servlet : Afin de pouvoir récupérer les informations pour une fonction donnée, et celle ayant besoin d'avoir certains paramètres, comme par exemple les informations pour une salle donnée, nous utilisons des liens situés dans les balises <a>.
  DBViewer
  En définissant le nom du paramètre à récupérer dans le servlet, on peut facilement lui donner des instructions. Il suffit de tester la valeur de ce paramètre avec un *IF* et lui effectuer l'affectation souhaitée.
  `String type = request.getParameter("choice");`

- Servlet vers JSP : Afin que l'on puisse créer du code dit "dynamique", le servlet se doit de passer les bons objets, remplis suite a l'interrgoation de la base MongoDB correspondantes, en paramètre à la page JSP afin qu'elle puissent être affichée. `request.setAttribute("prof", pr);`
  Il suffit ensuite d'appeler la bonne méthode dans le JSP en récupérant à son tour ce que le servlet nous retourne.
  `ArrayList pr = (ArrayList) request.getAttribute("prof");`

6 Problèmes rencontrés

Pour ce projet, nous avons bien sur rencontré quelques problèmes. Quelques-uns ont pu être résolu et d'autres non. La plus grosse difficulté pendant le développement de l'application est les requête de manipulation de la base de données MongoDB. En effet, contrairement au SGBD, nous manipulons des classes donc nous ne pouvions pas tout faire. Par exemple, avec SGBD, il est très facile de récupérer le nombre de champs d'une table et les noms de ces derniers (SELECT * ?.) mais avec MongoDB, nous ne pouvons pas faire cela. De plus, avec MongoDB, nous devions manipuler des objets, des classes et non pas la table directement. La deuxième grosse difficulté pour le coté administration réside dans le fait que nous ne connaissions pas bien MongoDB et ses requêtes, en effet, nous avons dû apprendre afin de connaitre et de maitriser les fonctions du MongoDB. Par rapport a ce qui a été énoncé dans le premier rapport, nous avons malheureusement pas pu être a la hauteur de nos ambitions. L'implémentation de 2 projets permettant de rendre l'utilité de cette application plus importante nous a pris énormément de temps. En effet parmi les 4 idées principales demandées dans l'énoncé, nous avons réussi à en implémenter 3 :

- Schémas et système dynamique : Création d'une interface graphique générique s'adaptant à l'évolution de la base.
- Interface d'interrogation : Création d'un champ permettant de filtrer les tuples de la base.
- Persistant des objets : Implémentation d'une base NoSQL, et stockages des objets dans l'application administrateur.

Par manque de temps nous n'avons pas pu approfondir certaines partie qui nous ont parue intéressante à améliorer. L'application sera améliorée d'ici a notre soutenance, afin de présenter un projet encore plus compétitif.

7 Conclusion

Afin de conclure ce rapport, nous nous sommes rendus compte que l'analyse préliminaire d'un projet, nous permet un gain de temps non négligeable, si les choix sont judicieusement élaborés. La création des différents diagrammes est essentielle à la réalisation d'une structure réfléchie et cohérente par rapport aux résultats que l'on veut obtenir.

Comme prévu dans le rapport précédent, certains de nos choix ont fait l'objet de discussions âpres. Comme vous le savez la mauvaise gestion du temps, et un mauvais cahier des taches a réaliser peuvent nous amener échouer un projet. Mais le rapport préliminaire a permis d'élaguer les mauvaises pistes, et de nous concentrer sur les fonctionnalités essentielles a la réussite de ce projet.

Ce projet, nous a paru très court du fait de notre grand intérêt pour la chose. C'est un projet qui nous a permis de nous ouvrir faire d'autres type de SGBD, nous faisant sortir des sentiers battus. Il est vrai que nous avons été déstabilisés au début, car après 5 ans de SQL, le NoSQL est difficile à assimiler. Mais nous avons su faire face et améliorer nos compétences dans le domaine de la base de données.

Nous sommes donc sorti grandi de ce projet, et la pluralité de notre projet a été bénéfique dans l'acquisition de nouvelles connaissances.