TER Base de répartition
2009/2010

Anne-Marie CAMARA, Mounia SEHILI,Mohamed SAKHI

Encadrés par:Olivier DALLE

Table des matières:

1-Introduction

Notre TER porte sur la base de répartition des enseignements du département informatique de l'Université Nice Sophia Antipolis.
Dans ce rapport nous commencerons par définir la problématique puis nous exposerons les outils et framework utilisés pour l'élaboration de ce travail. Dans une troisième partie nous parlerons de la conception et de l'implementation. Enfin pour finir nous exposerons les résultats et les problèmes rencontrés lors de ce TER

2-Problématique

Dans cette partie nous allons présenter ce qu'est une base de répartition et parler du pourquoi de ce projet.

2.1-Pourquoi du projet:

Le but du projet est d'intégrer à notre base certaines fonctionnalités manquantes dans la version précédente pour rendre le travail de l'administrateur le moins manuel possible et de corriger certaines erreurs, et surtout d'utiliser des outils modernes pour rendre cette base maintenable.
Parmi les problèmes de la version précédente on peut souligner le fait qu'on ne pouvait plus faire des mises à jour des différents outils ,ou de faire des opérations de maintenance car l'ancienne version a plus de 10ans et il y' aura certainement beaucoup d'incompatibilité , en plus de ca chaque fois qu'un utilisateur perd son mot de passe il doit contacter l'administrateur qui doit s'occuper de le chercher manuellement et de le lui envoyer ,donc nous avons intégré la fonctionnalité de la réinitialisation du mot de passe perdu à l'adresse email avec laquelle l'utilisateur a créé son compte.
Comme vous allez le constater dans ce qui suit nous avons choisi de travailler avec le framework Ruby on Rails.

2.2-Description de le base de répartition:

Une base de répartition est la plateforme qui *relie les enseignants et les enseignements*. C'est sur cette base que les *enseignants peuvent choisir d'assurer un cours* qui n'a pas encore trouvé preneur.
Il existe *différents types d'enseignants* comme les enseignants de l'université qui appartiennent au département, les enseignants de l'université qui n'appartiennent pas au département, les vacataires et d'autres types.

Pour les enseignants de l'université ils ont un nombre précis d'heure d'enseignements à assurer chaque année ,et si ils le dépassent ils seront payés pour des heures supplémentaires.
Il existe aussi *différents types de séances* et chaque type a son propre coefficient multiplicateur, comme les séances de cours, les séances de td et la combine des deux c'est à dire le type cours/td.
Tous les utilisateurs de la base n'ont pas *les mêmes droits* pour les différentes fonctionnalités qu'offre cette base.
Le but d'avoir une base de répartition est de connaître les enseignement n'ayant pas pris preneur et donc se proposer pour un tel enseignement, et aussi de faire le bilan des enseignements ou d'une filière.

3-Technologies:

3.1-Les frameworks proposés:

Pour réaliser ce travail on avait le choix entre plusieurs frameworks mais notre encadrant nous a proposé de tester seulement Ruby on rails et python+ django puisqu'ils sont les plus utilisés sur le marché et qu'il y 'a une grande communauté de développeurs qui travaille sur leur développement et leur maintenance.

3.1.1Django:

Django est un framework qui s'inspire du principe MTV (la vue est gérée par un template) composé de 3 parties distinctes:

1.Un langage de template flexible qui permet de générer du HTML,XMLou tout autre format texte;
2.Un contrôleur fourni sous la forme d'un "remapping" d'URL à base d'expressions rationnelles;
3.Une API html d'accès aux données est automatiquement générée par le framework compatible CRUD. Inutile d'écrire des requêtes SQL associées à des formulaires, les requêtes sql sont générées automatiquement par l' ORM.
En plus de l'API d'accès aux données, une interface d'administration fonctionnelle est générée depuis le modèle de données. Un système de validation des données entrée par l'utilisateur est également disponible et permet d'afficher des messages d'erreurs automatiques.

Sont également inclus:

- Un serveur web léger permettant de développer et tester ses applications en temps réel sans déploiement.

- Un système élaboré de traitement des formulaires muni de Widgets permettant d'interagir entre du HTML et une base de données. De nombreuses possibilités de contrôles et de traitements sont fournies.
- Un framework de cache web pouvant utiliser différentes méthodes (MemCached, système de fichier, base de données, personnalisé).
- Le support de classes intermédiaires (middleware) qui peuvent être placées à des stades variés du traitement des requêtes pour intégrer des traitements particuliers (cache, internationalisation, accès...)
- Un support complet d'Unicode

3.1.2-Ruby on rails

Ruby on rails est un framework web libre écrit en Ruby. Il suit le motif de conception « modèles-vue-controlleurs » aussi nommé MVC. Il permet de créer des applications web rapidement, car il impose une structure au programmeur, et ainsi l'oblige à avoir une logique et une démarche qui favorise la réalisation de l'application.

3.1.3-Choix du framework:

Aprés trois semaines de tests on a décidé de travailler avec Ruby on Rails puisqu'avec ce framework:

-on peut developper beaucoup plus rapidement.

-Ruby est langage simple.

-plus de fonctionnalités sous forme de plugins.

-grande souplesse pour faire des modifications sur la base de donnée.

-Moins de code à produire, c'est donc moins de temps à coder, moins d'erreurs potentielles, moins de code à maintenir.

-structuration du développement car il repose sur le MVC qui sépare l'application entre les données, la présentation et le traitement

-Utilisation de REST.

3.2-SVN

SVN ou subversion est un logiciel qui permet de gérer les différentes versions du code écrit. Pour notre application nous avons choisi de travailler avec la forge de google code qui nous permettait de récupérer des anciennes versions tout en soulignant les parties de code modifiées entre deux versions. IL permet aussi le renommage et le déplacement de fichiers ou de répertoires sans en perdre l'historique,et on peut attacher des propriétés, comme les permissions, à un fichier.

3.3-Machine virtuelle

Pour notre projet on a travaillé sur la machine virtuelle qui se trouve sur largo.unice.fr pour éviter tout problème de conflit de version puisque notre application va être déployée sur cette même machine, ainsi que pour son accessibilité partout via les tunnels SSH.

4-Description du framework choisi

4.1-L'architecture Modèle-vue-controlleur

Rails fournit des outils pour construire une architecture MVC:

4.1.1-Les modèles

Ce sont les classes assurant la gestion des données. En général la structure de ces classes est déterminée automatiquement par Rails à partir d'une base de données. Les relations entre les tables sont explicitement spécifiées (*has_many belongs_to*). Spécifier ces relations permet à ActiveRecord de précharger des éléments de classes enfants ou parent.

4.1.2-Les vues

Elles déterminent comment sont affichées les informations à l'utilisateur. Il s'agit généralement d'une combinaison de code **Html** et de **Ruby** dans des fichiers *.html.erb*.

4.1.3-Les contrôleurs

Ils réagissent aux actions des utilisateurs, ils vont chercher les données dans la base et les mettent à disposition aux vues.

4.2-Fonctionnement de ruby on Rails:

-Création de l'application

Il suffit juste de tapper la commande rails myapp

-Création de la base de donnée:

elle se fait par la commande :

myapp> rake db:create

-Création des modèles, des vues et des contrôleurs:

modèle: c'est par la commande

myapp>ruby/script model user

le contrôleur :

myapp>ruby/script controller user

la vue: il existe différentes vues selon les méthodes implémentées dans le contrôleur c'est à dire on doit associer à chaque méthode une vue.

4-Sauvegarde des fichiers et des tables:

par cette commande:

myapp>rake db:migrate

5-Ajout de fonctionnalités:

Si on oublie quelques champs dans les tables, on peut les rajouter facilement avec ruby on rails. Il suffit de faire un rollback par la commande:

myapp> rake db:rollback

et après faire les modifications(ajouts de nouveaux champs, suppression, ..etc) ensuite il faut enregistrer ces modifications avec la commande:

myapp> rake db:migrate

4.3-Les répertoires générés par rails:

Après avoir créé une application rails, des répertoires sont générés ce qui structure l'application, organise et facilite le travail du développeur.

- **app :** Ce répertoire organise les composants de l'application: les modèles, les vues et les contrôleurs.
- **app/controllers:** les contrôleurs c'est là ou rails cherche à trouver les classes du controlleur. Un controlleur gère les demandes web de l'utilisateur.
- **app/view:** contient les vues de l 'application.
- **app/models:** Les modèles contiennent les classes et les données stockées dans la base de notre application.
- **config:** Ce répertoire contient la petite quantité de code de configuration que l' application aura besoin, y compris la configuration de la base de données (en database.yml), la structure de l'environnement Rails (environment.rb), et le routage des requêtes Web entrantes (routes.rb).
- **db:** contient un sous repertoire 'migrate' qui contient toutes les tables de la base de donnée et un fichier schema.rb qui decrit toutes les tables
- **doc:** Ruby a un cadre, appelé RubyDoc, qui peuvent générer automatiquement la documentation pour le code qu'on crée.
- **Lib:** Contient toutes les bibliothèques, à moins qu'ils appartiennent explicitement ailleurs.
- **Public:** Ce répertoire contient des fichiers Web qui ne changent pas, comme les fichiers JavaScript (public / javascripts), graphiques (public / images), les feuilles de style (public /

stylesheets), et les fichiers HTML (public).

- **Script:** Ce répertoire contient des scripts pour lancer et gérer les différents outils qu'on utilise avec Rails. Par exemple, il existe des scripts pour générer du code (générer) et de lancer le serveur web (serveur).
- **Readme:** Ce fichier contient un détail de base sur Rail.
- **Rakefile:** Ce fichier est similaire à Unix Makefile qui contribue à la construction, l'emballage et les tests du code Rails.

5-ANALYSE ET CONCEPTION

Pour la conception et la réalisation de notre application, nous avons choisi de modéliser avec le formalisme UML (Unified Modeling Language) qui offre une flexibilité marquante qui s'exprime par l'utilisation des diagrammes.

5.1-Spécification des besoins

La plupart des nouveaux langages sont orientés objet. Le passage de la programmation fonctionnelle à l'orienté objet n'était pas facile. L'un des soucis était d'avoir une idée globale en avance de ce qu'on doit programmer.

L'algorithmique qui était utilisé dans la programmation fonctionnelle ne pourrait pas suffire à lui seul. Le besoin d'avoir des méthodes ou langages pour la modélisation des langages orientés objet se faisait sentir. Ainsi plusieurs méthodes ou langages on vu le jour. En occurrence UML qui nous a permis de faire la conception de notre application.

De nos jours UML2 possède treize diagrammes qui sont classés en deux catégories (dynamique et statique).

Pour ce faire on a commencé par les diagrammes de cas d'utilisation (Use Case) qui permettent de donner une vue globale de l'application. Pas seulement pour un client non avisé qui aura l'idée de sa future application mais aussi le développeur s'en sert pour le développement des interfaces.

En deuxième lieu on va présenter la chronologie de quelques opérations par les diagrammes de séquences. Et enfin le diagramme des classes.

5.2- Diagramme des cas d'utilisation

Le but de ces diagrammes et d'avoir une vision globale sur les interfaces du futur logiciel. Ces diagrammes sont constitués d'un ensemble d'acteurs qui agit sur des cas d'utilisation.

UML n'emploie pas le terme d'Utilisateur mais d'acteur. Les acteurs d'un système sont les entités externes à ce système qui interagissent avec lui. Suivant les besoins de notre système on peut présenter deux acteurs Il s'agit d'un administrateur et un intervenant au sein du département informatique . La manière d'accéder aux services de l'application pour l'un et pour l'autre est la même. La différence réside sur les droits d'accès et les limites de chacun.

comme l'administrateur se place en dessus de l'intervenant celui-ci peut faire à part les taches de l'intervenant mais aussi gérer ces derniers. Ceci est intéressant car UML présente le critère d'héritage entre les acteurs. Donc on pourra faire l'administrateur un héritier de l'intervenant et on va lui ajouter la particularité de pouvoir gérer des intervenants.

5.2.1 Cas d'utilisation de l'intervenant

.Un intervenant peut soit s'authentifier, soit se porter candidat sur une séance, ou sur un cours . Il peut aussi savoir ou il en visualisant la listes des cours, la listes des candidatures)

. Aussi, selon ses responsabilités un intervenant pourra affecter un cours, une séance. Ainsi un responsable de cours pourra affecter des cours et faire des mises à jour sur
les cours, un responsable de séances pourra lui aussi affecter des séances et faire des mises à jour sur la table séance.

« Diagramme des cas d'utilisation d'un intervenant »

5.2.2 Cas d'utilisation de l'administrateur

Un administrateur peut soit s'authentifier, soit donner des permissions. Il peut aussi effectuer des mises à jour et affecter des cours à un intervenant.

5.3-

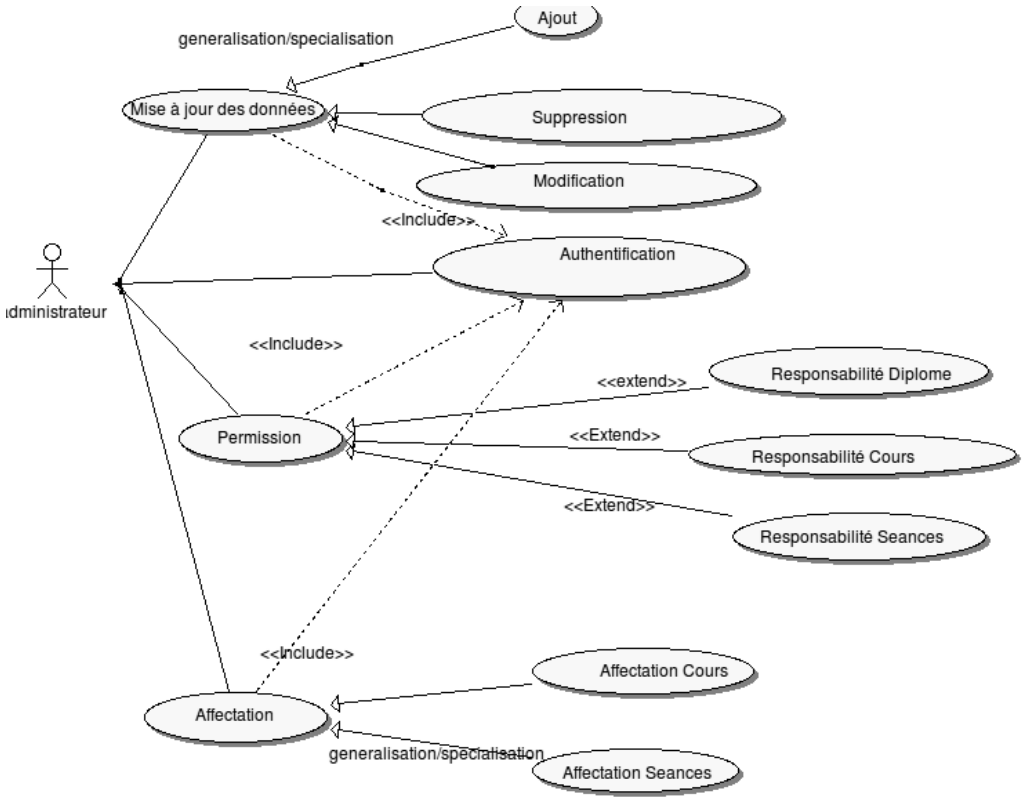
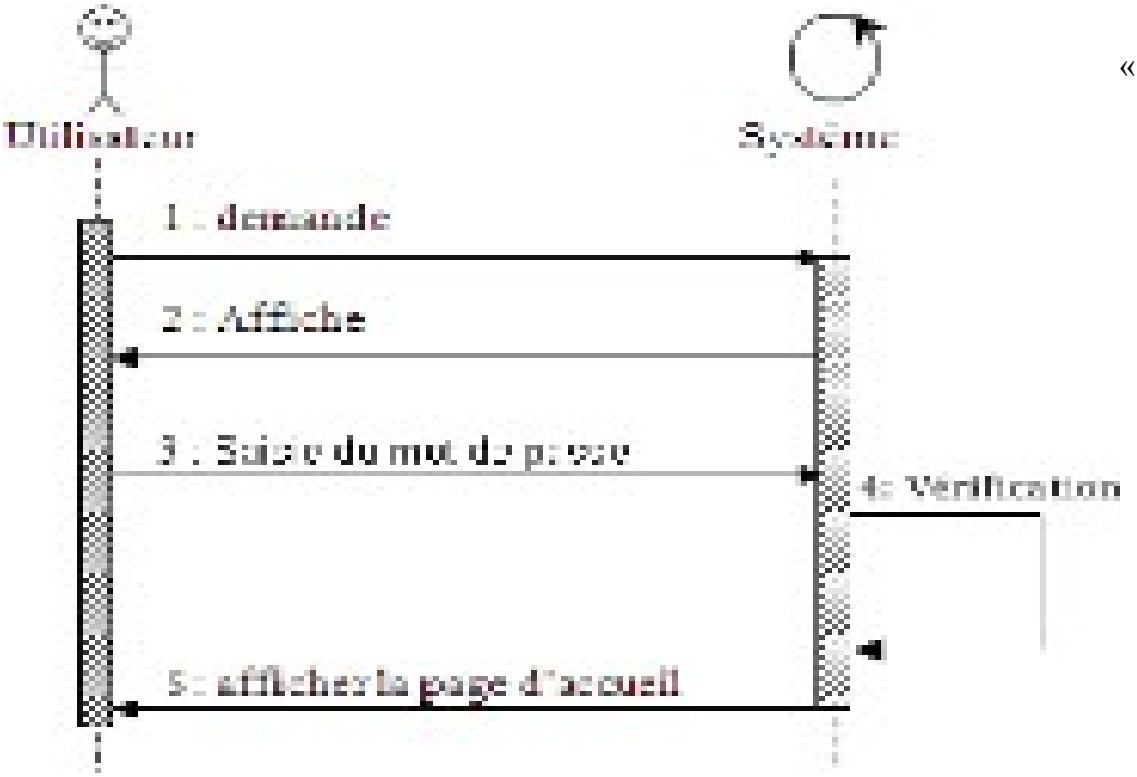


DIAGRAMME DE SEQUENCE

Voici les diagramme de séquence pour l'authentification et de l'ajout

5.3.1Authentification

1. l'utilisateur demande le formulaire d'authentification.
2. L'application affiche le formulaire d'authentification.
3. L'utilisateur saisit le mot de passe.
4. Le système vérifie la validité du mot de passe.
5. L'application affiche la page d'accueil.



« Diagramme de *séquence de l'authentification »*

5.3.2 Ajout

1. L'utilisateur demande le formulaire d'ajout.
2. L'application affiche le formulaire d'ajout.
3. L'utilisateur saisit les nouvelles données.
4. L'application envoi la requête.
5. L'application stocke les données au niveau de la base de données.
6. L'application confirme l'enregistrement.

5.4-

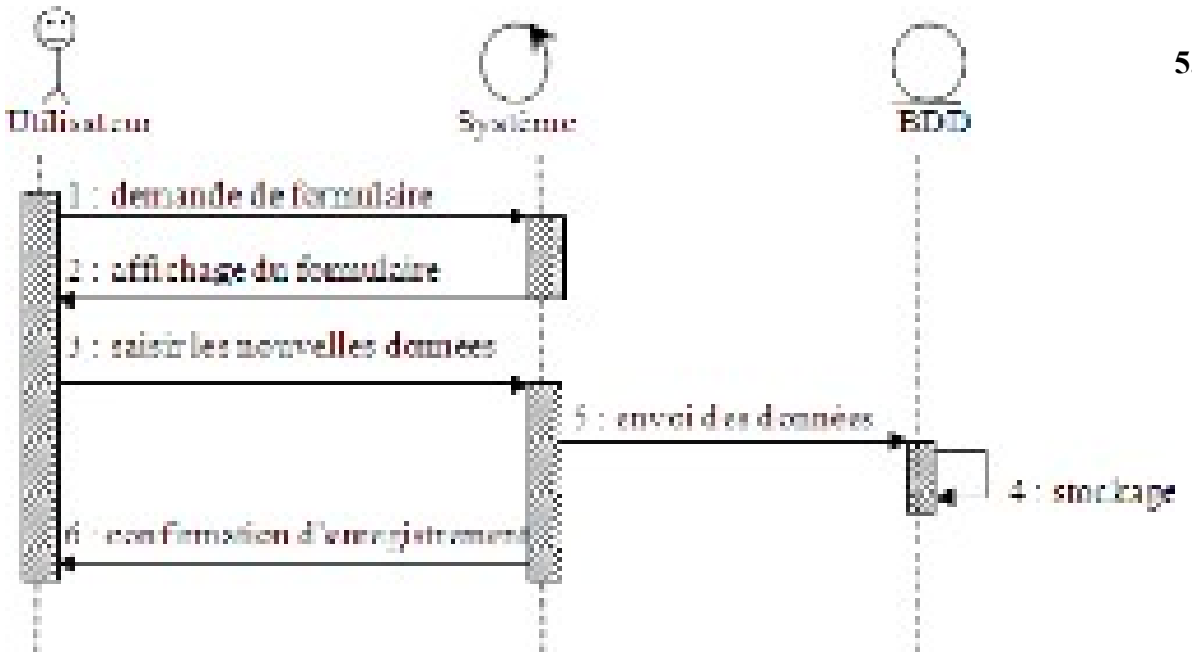
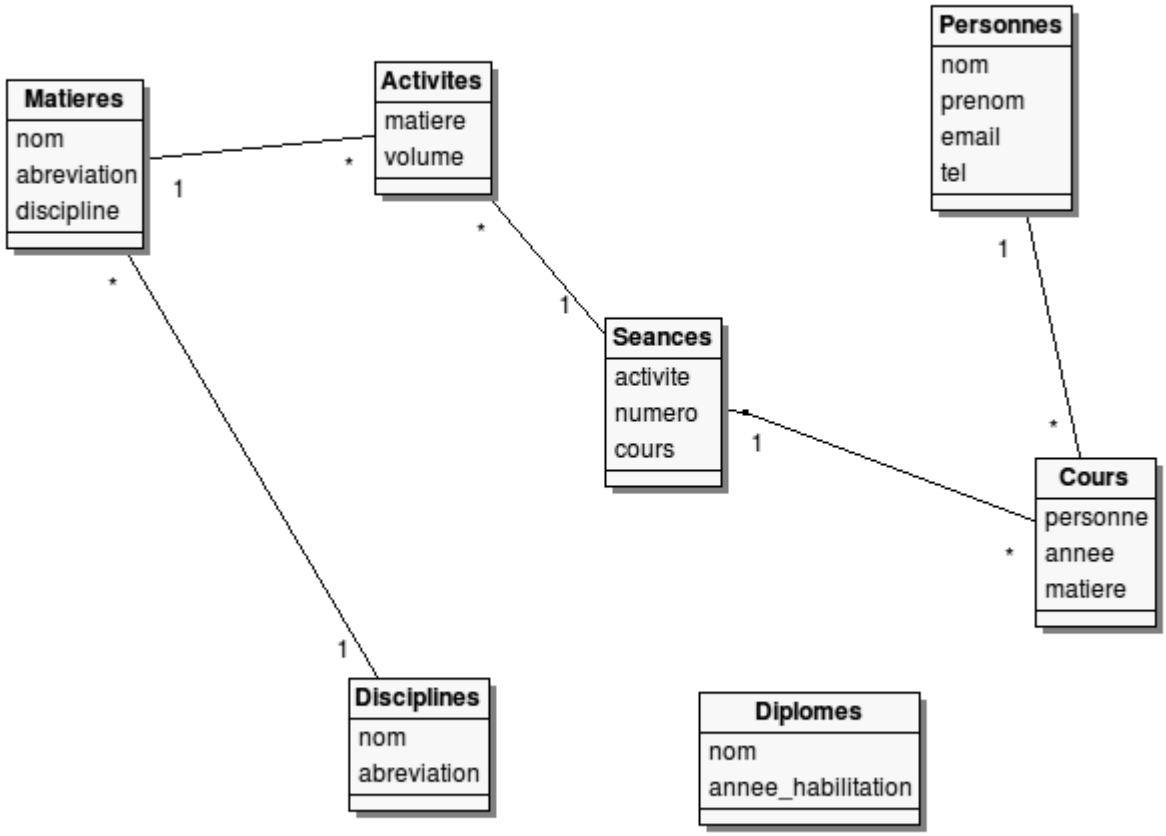


Diagramme des classes

Ce modèle nous permet d'avoir une vue statique de l'application. Il nous montre les relations entre les différentes entités (classes) composant notre application. Il nous mène vers la solution finale. À partir de ce diagramme on retrouve les corps des différentes classes de notre application.



« Diagramme des classes »

6-Implémentation:

6.1-Plugin d'authentification

Ces plugins proposent les solutions d'authetification suivantes:

-Login/logout
-Manipulation sécurisée des mots de passe

-Activation de compte en envoyant un email d'activation

-Validation et invalidation de compte par l'administrateur.

-Autorisations et contrôle d'accés

On a dû tester la plupart des plugins disponibles sur le marché ,qui sont:

6.1.1Authlogic:

Ce plugin propose une solution d'authentification simple, propre et discrète.Authlogic propose un nouveau type de modèle.On peut avoir tant qu'on veut de modèles et les nommer comme on veut.Authlogic crée un UserSessionController et l'utilise comme tous les autres modèles.Il fait l'authentification en premier puis envoi les informations et les cookies de la session.

6.1.2-Restful

Ce plugin largement utilisé fournit une base pour un management sécurisé de l'authentification des utilisateurs. Ce plugin est différent des acts_as_authenticated en générant deux contrôleurs et exige des routes différentes.

6.1.3-Clearance

Ce plugin se concentre sur la maitenabilité du code des opérations d'authentification.Il encapsule l'authentification dans des modules inclus dans les controleurs.Son approche maintient le code de notre application rails propre et nous alerte au cas d'infraction de ces règles

6.1.4-choix du plugin

On a choisi plugin d'Authlogic puisqu' aprés les tests qu'on a fait sur la machine virtuelle c'était le seul qui permettait de réaliser les opérations demandées telles que l'envoi du mot de passe perdu .

6.2-Plugin de l'interface d'administration

Ce plugin permet à l'administrateur d'avoir sa propre interface d'administration ou il peut gérer toutes les données de l'administration.On avait le choix entre plusieurs plugins parmis eux:

6.2.1-Typus

Ce plugin a une belle interface d'administration et facile à configurer.On peut configurer quelles colonnes on désire affiche et lesquelles sont cherchées.Il est maintenu par quelques contributeurs et un google group enthousiaste

6.2.2-Admin data

Ce plugin propose aussi des options de gestion de l'interface d'administration tout en intègrant aussi la notion d'authetification en ajoutant quelques lignes de code dans l'initialiseur situé dans: /config/initializers/admin_data.rb

6.2.3-Hobo

Ce plugin fait dans la gestion par défaut des droit d'accès aux bases de données rendant le paramétrage d'un prototype aussi simple que le "scaffold" de Rails mais avec plus de paramètres prédéfinis, l'édition des données dans l'application qui intègre des composants AJAX par défaut et pour finir la gestion de plusieurs thèmes par l'application web. Un système de tagging, le DRYML, permet aussi de limiter l'intégration de code dans les vues et de les remplacer par un tag défini qui se verra remplacé lors de l'exécution de l'application.

6.2.4-Streamlined

Ce plugin n'est pas compatible avec la version 2.3 de Rails

On a choisit de travailler avec typus parce qu'il offre beaucoup plus de fonctionnalités, et c'était le premier qu on a bien pu intégrer dans notre application située sur la machine virtuelle

6.2.5-Choix du plugin

Pour des raisons de temps on a pas pu tester tous ces plugins vu qu'avec typus on a pu mettre en place une interface d'administration et on n'avait pas de soucis à l'intégrer à notre application hébergée sur la machine virtuelle

6.3 Mise en Place:

6.3.1- Inscription des nouveaux utilisateurs:

-Installation de la gem authlogic.

-Création du modèle user session.

-Création du modèle user.

-Edition du fichier create user.rb qui se trouve dans db/migrate pour ajouter les colonnes néssaires au foncionnement d'authlogic Voici un exemple:

```
  class CreateUsers < ActiveRecord::Migration

      def self.up
        create_table :users do |t|
          t.string :login, :null => false
          t.string :email, :null => false
          t.string :crypted_password, :null => false
          t.string :password_salt, :null => false
          t.string :persistence_token, :null => false
          t.string :perishable_token, :null => false
          t.integer :login_count, :null => false, :default => 0
          t.integer :failed_login_count, :null => false, :default
=> 0
          t.datetime :last_request_at
          t.datetime :current_login_at
          t.datetime :last_login_at
```

```
      t.string :current_login_ip
      t.string :last_login_ip
      t.timestamps
    end
  end

  def self.down
    drop_table :users
  end
end
```

- **crypted_password et password_salt** sont utilisés pour le cryptage et le stockage du mot de passe
- **persistence_token** sert à maintenir l'utilisateur connecté
- **perishable_token** est utilisé pour identifier un utilisateur lors de la confirmation de son ouverture de compte et de la réinitialisation de son mot de passe.
- **login_count** sert à stocker le nombre de fois où l'utilisateur s'est connecté
- **failed_login_count** est utilisé pour la protection contre les attaques "brute force" (par défaut Authlogic suspend un utilisateur pour 2 heures au bout de 50 essais infructueux, ces valeurs étant bien sûr configurables.)
- **last_request_at** sert à stocker la date de dernière action sur le site
- **current_login_at** sert à stocker la date de dernière connexion
- **last_login_at** sert à stocker la valeur de **current_login_at** avant qu'il ne soit réinitialisé
- **current_login_ip** sert à stocker l'adresse IP de dernière connexion
- **last_login_ip** sert à stocker la valeur de **current_login_ip** avant qu'il ne soit réinitialisé

-mise à jour la base de donnée avec la migration en tappant la commande: « rake db:migrate »

-Création des controlleurs pour permettre aux nouveaux utilisateurs de s'inscrire.

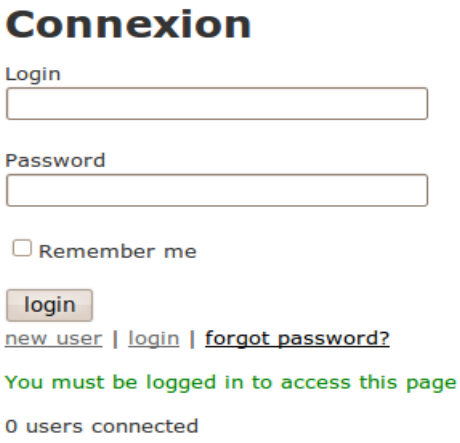
-Création des vues correspondantes.

6.3. 2-Connexion et Déconnexion des utilisateurs:

Une fois l'utilisateur créé, nous devons lui permettre de se connecter. Nous commençons par créer une page d'accueil sur laquelle nous verrons :

-un lien vers le formulaire de connexion si l'utilisateur n'est pas connecté.

-login de l'utilisateur connecté.



Pour ce faire:

-Création du controlleur user_session.

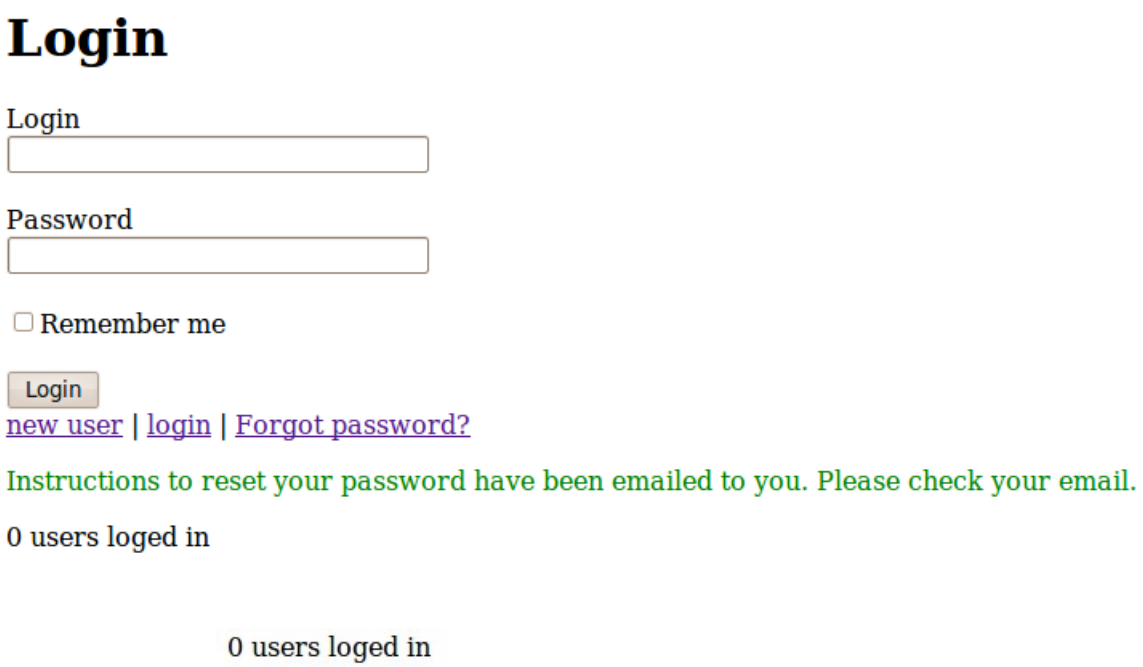
-Création des vues correspondantes.

6.3.3-Réinitialisation des mots de passes perdus:

Il arrive fréquemment qu'un utilisateur oublie son mot de passe. Nous allons lui permettre de le réinitialiser de manière simple et sûre grâce à Authlogic.

Le processus de réinitialisation d'un mot de passe suit les étapes suivantes :

1. l'utilisateur demande la réinitialisation de son mot de passe
2. un email contenant un lien de réinitialisation lui est envoyé
3. l'utilisateur clique sur le lien et arrive sur un formulaire où saisir son nouveau mot de passe
4. une fois le mot de passe modifié l'utilisateur est automatiquement connecté et le lien ayant servi à la réinitialisation est expiré



« capture d'écran pour un formulaire de réinitialisation du mot de passe »

Nous allons avoir besoin de deux champs **email** et **perishable_token** sur le modèle **User**

-Nous commençons par une migration:

« add_user_password_reset_fields »

-Nous ajoutons les champs nécessaire.

-Nous appliquons la migration.

-nous créons le controlleur password_reset_controller avec les deux méthodes: create et new qui cherchent un utilsateur avec son mail

« find by email »

-Nous définissons une méthode « deliver_password_reset_instructions » qui envoie un mail contenant le lien de réinitialisation.

Voici un exemple:

```
class User < ActiveRecord::Base
     def deliver_password_reset_instructions!
```

```
        reset_perishable_token!
        UserMailer.deliver_password_reset_instructions(self)
      end
    end
```

la méthode reset_perishable token génère un nouveau perishable token et sauvegarde l'enregistrement.

-Nous créons le mailer qui contient le message

-Nous créons les vues correspondantes.

-Nous créons le controlleur: passeword_reset_controller qui contient essentiellement cette méthode:

« load_user_using_perishable_token » permet de retrouver l'utilisateur à partir du perishable token contenu dans le lien. Elle utilise la méthode « find_using_perishable_token » fournie par authlogic. Elle assure que le token utilisé n'est pas vide et est toujours valide parce que authlogic expire automatiquement les tokens au bout des 10 minutes.

Si la modification est faite, l'utilisateur est automatiquement connecté grace à authlogic.

6.4ActiveRecord, modèles et migrations

ActiveRecord est la couche ORM utilisée dans Rails

Il est utilisé par les controleurs comme proxy pour les tables de base de données. Le concept de base de ActiveRecord est le modèle. Chaque classe du modèle est stockée dans le répertoire app/models de l'application.
Chaque modèle correspond habituellement à une table dans la base de données.
Dans le cas de notre application, les modèles sont organisées comme suit:

Table	**classe du modèle**	**Fichier contenant la classe de définition (dans app / models)**
activites	Activite	activite.rb
cours	Cour	cour.rb
diplomes	Diplome	diplôme.rb
discplines	Discipline	discipline.rb
matieres	Matiere	matiere.rb
personnes	Personne	personne.rb
seances	Seance	seance.rb
users	User	user.rb

Les « migrations » dans Rails permettent de générer une structure de la base de données en utilisant une série de fichier de script Ruby pour définir les opérations de base de données. Aussi les migrations ne sont pas seulement utilisées pour la création des tables, mais permettent aussi la suppression des tables, leur modification, et même l'ajout des données dans les tables.

Lors de l'éxécution d'une migration, le script Ruby est converti en SQL appoprié au serveur de base de données et est éxécuté sur la connexion de la base.

7-Répartition des tâches:

Pour réaliser cette application, notre travail était divisé en 2 parties:

7.1-Une phase de test:

Dans cette partie:

Un d'entre nous s'est occupé de faire des tests sur Ruby on Rails en créant une petite application avec un minimum de tables et essayait de se documenter et de chercher les différents plugins pour les différentes fonctionnalités

L'autre s'est occupé de se documenter et de créer les tables en python sur Django tout en essayant de créer et adapter l'interface d'administration.

Un 3ème s'occupait de chercher et de tester les différents plugins pour les fonctionnalités de récupération de mot de passe et aussi de confirmation par e-mail.

7.2 -Une phase d'implémentation:

Aprés avoir choisi le framework et puisqu'on était déjà en retard sur notre application, on s'est mis d'accord que chacun d'entre nous s'occupe de tester un plugin parmi ceux cités précédemment pour la récupération de mot de passe par mail, alors qu'en ce moment un d'entre nous s'est occupé de créer les tables et lesrelations entre elles et enfin l'autre essayait de travailler sur le plugin de l'interface d'administration.

8-Conclusion

Pendant cette période de TER nous avons connu et manipuler différents outils tels que SVN, SSH. Et surtout nous avons connu deux frameworks qui sont très puissant et très demandés sur le marché du travail.
Nous avons aussi appris comment gérer les problèmes rencontrés et leur trouver les solutions adéquates. N'oublions pas de mentionner qu'on a appris à travailler en groupe et gérer le temps qui nous a été confié pour réaliser notre problème ce qui a été l'un de nos plus grands soucis.

8.1-Ce qui a été fait:

Arrivés à terme de la durée qui nous a été confiée pour réaliser notre application, nous sommes arrivés à créer les différentes tables et les relations entre elles tout en mettant en place la fonctionnalité de récupération des mots de passe perdus grâce à un email que reçoit l'utilisateur à l'adresse qu' il a indiqué lors de la création de son compte utilisateur. Enfin nous avons aussi ajouté un plugin pour l'interface d'administration qui permet à l'administrateur d'effectuer les différentes opérations d'administration.

8.2-Problèmes rencontrés

Les difficultés majeures rencontrées lors de ce TER sont les suivantes:
-Problème de version sur la machine virtuelle largo
-Panne du serveur
-Mauvaise gestion du temps
-Une documentation pas assez fournie de rails.

8.3Ce qui reste à faire

Vu qu'on a pris beaucoup de retard pour choisir le framework avec lequel on va travailler il reste encore quelques améliorations à apporter à notre application notamment la création de formulaires qui font les opérations suivantes:
-Affectation d'un responsable pédagogique.
-Affectation des cours par le responsable pédagogique.
-Découpage en groupes par le responsable pédagogique.
-Affectation des séances par le responsable pédagogique.
-Affectation d'une séance à une autre personne si le responsable du cours ne peut l'assurer.

9-Bibliographies:

http://www.stoneageblog.com/articles/2007/03/04/ma-premiere-application-ruby-on-rails/

http://fr.wikipedia.org/wiki/Ruby_on_Rails

http://www.binarylogic.com/2008/11/16/tutorial-reset-passwords-with-authlogic/

http://www.tutoriaux-rails.com/articles/inscription-et-connexion-d-un-utilisateur-avec-authlogic/

http://docs.django-fr.org/intro/tutorial01.html

http://rubyonrails.org/

http://wiki.rubyonrails.org/