Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen

Faculté des Sciences

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Licence en Informatique

Thème

Conception et réalisation d'un Système de Gestion de Demande de Formation pour une Entreprise GDFE

Réalisé par :

- **MAHI Anes.**

Encadré par :

- **SMAHI Mohammed Ismail.**

Présenté le 30/06/2011 devant la commission d'examination composée de MM.

- *Midouni Djallal* *(Examinateur)*
- *El Yebdri Zeyneb* *(Examinatrice)*
- *Halfaoui Amel* *(Examinatrice)*

Année universitaire: 2010-2011

Remerciement

En témoignage de ma gratitude et de mon profond respect jôadresse mes vifs et sincères remerciements à :

Dieu miséricordieux, pour son aide et sa bénédiction.

A mon encadreur monsieur SMAHI pour ses conseils, sa disponibilité et son encouragement qui môont permis de réaliser ce travail dans les meilleures conditions.

Les jurys pour leurs efforts et leur soin apporté à notre travail.

Aux enseignants de notre université et département informatique.

Enfin, je tien à dire que le soutien quotidien de mes parents môa été très utile.

Table des matières

Table des illustrations

Introduction générale

Contexte

Lœentreprise actuelle favorise lœapprentissage par internet et offre la possibilité à ses employés et aux différents apprenants souhaitant lœintégration de se former dans le domaine désiré, en mettant en leur disponibilité un catalogue de formations classées par thème et possédant plusieurs sessions afin de suivre lœactualité.

Objectifs

Lœintérêt de ce travail et de créer un catalogue de formation au service des employés de lœentreprise et aux différents internautes pour se former dans de multiples domaines. Pour cela nous allons concevoir un système de gestion de demande de formation baptisé GDFE.

Plan du document

Le document est organisé en chapitres. Le découpage a été fait de manière à couvrir les domaines impliqués dans notre travail.

Le **chapitre 1** est consacré à la présentation UML (les définitions, les descriptions, et lœutilité des différents digrammes).

Le **chapitre 2** est consacré à lœétude de notre système en présentant les digrammes montrant la conception du projet.

Le **chapitre 3** est la partie réalisation et implémentation du logiciel, où nous allons expliquer le choix du langage de programmation, en décrivant dœune manière général lœapplication réalisée via une étude de cas.

Chapitre 1 : Le Langage de modélisation UML

I. Introduction [1]

Le Génie logiciel et la méthodologie sôefforcent de couvrir tous les aspects de la vie du logiciel. Issus de lôexpérience des développeurs, concepteurs et chefs de projets, ils sont en constante évolution, parallèlement à lôévolution des techniques informatiques et du savoir-faire des équipes.

Comme toutes les tentatives de mise à plat dôune expérience et dôun savoir-faire, les méthodologies ont parfois souffert dôune formalisation excessive, imposant aux développeurs des contraintes parfois contre-productives sur leur façon de travailler.

Avec la mise en commun de lôexpérience et la maturation des savoir-faire, on voit se développer à présent des méthodes de travail à la fois plus proches de la pratique réelle des experts et moins contraignantes.

UML, qui se veut un instrument de capitalisation des savoir-faire puisquôil propose un langage qui soit commun à tous les experts du logiciel, va dans le sens de cet assouplissement des contraintes méthodologiques.

II. Historique [2]

La programmation orientée objet consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle *domaine*) en un ensemble d'entités informatiques. Ces entités informatiques sont appelées *objet*. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, la couleur, ...).

L'approche objet est une idée qui a désormais fait ses preuves. Simula a été le premier langage de programmation à implémenter le concept de classes en 1967 ! En 1976, Smalltalk implémente les concepts d'encapsulation, d'agrégation, et d'héritage (les principaux concepts de l'approche objet). D'autre part, de nombreux langages orientés objets ont été mis au point dans un but universitaire (Eiffel, Objective C, Loops...). [1]

Un objet représente une entité du monde réel, ou de monde virtuel dans le cas dôobjets immatériels, qui se caractérisent par une identité, des états significatifs et par un comportement.

L'identité d'un objet permet de distinguer les objets les uns par rapport aux autres. Son état correspond aux valeurs de tous les attributs à un instant donné. Ces propriétés sont définies dans la classe d'appartenance de l'objet.
Enfin, le comportement d'un objet se défini par l'ensemble des opérations qu'il peut exécuter en réaction aux messages envoyés (un message = demande d'exécution d'une opération) par les autres objets. Ces opérations sont définies dans la classe d'appartenance de l'objet.

La modélisation objet consiste à créer une représentation informatique, des éléments du monde réel, aux quels on s'intéresse sans se préoccuper de l'implémentation, ce qui signifie indépendamment d'un langage de programmation. Il s'agit donc de déterminer les objets présents et d'isoler leurs données et les fonctions qui les utilisent. Pour cela des méthodes ont été mises au point entre les années 70 et 90, de nombreux Analystes ont mis au point des approches orientées objets, si bien qu'en 1994, il existait plus de 50 méthodes objet. A partir de cette année, Rumbaugh et Booch (rejoints en 1995 par Jacobson) ont unis leurs efforts, pour mettre au point la méthode unifiée (Unified Method 0.8), incorporant les avantages de chacune des méthodes précédentes. La méthode unifiée à partir de la version 1.0 devient UML, soumis à l'OMG (Object Management Group) en Janvier 1997, et acceptée en novembre 1997 dans sa version 1.1, date à partir de la quelle UML devient un standard international. La version qui à vu nos jours est la version 2.0 d'UML.

III. Définition UML

UML (en anglais Unified Modeling Language ou « langage de modélisation unifié ») est un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.
Il est apparu dans le monde du génie logiciel, dans le cadre de la « conception orientée objet ». Couramment utilisé dans les projets logiciels, il peut être appliqué à toutes sortes de systèmes ne se limitant pas au domaine informatique.

UML cadre l'analyse objet, en offrant :

- → Différentes vues (perspectives) complémentaires d'un système, qui guide l'utilisation des concepts objets.

- → Plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.

UML est un support de communication

- → Sa notation graphique permet d'exprimer visuellement une solution objet
- → L'aspect formel de sa notation limite les ambiguïtés et les incompréhensions
- → Son aspect visuel facilite la comparaison et l'évaluation de solutions

Son indépendance (par rapport aux langages d'implémentation, domaine d'application, processus) en font un langage universel.

Les points forts d'UML [3]

- → UML est un langage formel et normalisé, il permet un gain de précision et un gage de stabilité, ce qui encourage l'utilisation des outils.
- → UML est un support de communication performant, il cadre l'analyse et facilite la compréhension de représentation abstraites complexes.
- → Son caractère polyvalent et sa souplesse en font un langage universel.

Les points faibles d'UML

- → Le mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.
- → UML n'est pas à l'origine des concepts objets, mais en constitue une étape majeur, car il unifie les différentes approches et en donne une définition plus formelle.
- → Le processus (non couvert par UML) est une autre clé de la réussite d'un projet. Or, l'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est une tâche complexe et longue.

IV. Les éléments UML

IV.1. Les éléments structurels

IV.1.1 Classe

C'est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés. Elle peut être privée, publique ou protégée.

Nom classe
Attributs
Méthodes

IV.1.2 Interface

La principale utilisation des interfaces est la modélisation des points de soudure au sein dôun système constitué de composants logiciels.

IV.1.3 Cas dôutilisation

Un cas dôutilisation saisit le comportement attendu du système.

IV.1.4 Collaboration

Cêst un ensemble de classes, dôinterface et dôautres éléments qui travaillent ensemble pour fournir un comportement de coopération.

IV.1.5 Classe active

Cêst une classe dont les objets possèdent un ou plusieurs processus ou threads et qui peut donc lancer une activité de commande, elle ressemble à une classe sauf que ses objets représentent des éléments dont le comportement est courent.

IV.1.6 Composant

Cêst une partie physique remplaçable dôun système qui se conforme à un ensemble dôinterfaces et qui permet la réalisation.

IV.1.7 Nî ud

Cêst un élément physique qui intervient lors de la phase dêexécution, il représente une ressource de calcul et dispose généralement dôun peu de mémoire.

IV.2. Les éléments de regroupement

Représente les parties organisationnelles des modèles UML.

IV.3. Les éléments comportementaux

Ce sont les verbes du modèle et représentent leur comportement dans le temps et lêspace.

IV.3.1 Relation

Cêst une liaison entre objets (classes) en modélisation orienté objet.

IV.3.2 Dépendance

Cêst une relation dêétatisation qui indique quôun changement de spécification dôun élément peut affecter un autre élément qui lôutilise, mais lôinverse nêst pas vrai.

IV.3.3 Généralisation

Cêst une relation entre un élément générale appelé super classe ou parent et une version plus spécifique appelée classe fille ou sous classe.

IV.3.4 Association

Cêst une relation structurelle qui indique que les objets dûun élément sont liés à des objets dûun autre élément. En reliant deux classes, elle autorise la navigation dûun objet de lûune dêelles vers un objet de lêautre et vice versa.

V. Vues et diagrammes UML [4]

On commence par définir ce quûune vue et ce quûun diagramme

- Les ***vues*** : Les vues sont les observables du système. Elles décrivent le système d'un point de vue donné, qui peut être organisationnel, dynamique, temporel, architectural, géographique, logique, etc. En combinant toutes ces vues, il est possible de définir (ou retrouver) le système complet.
- Les ***diagrammes*** : Les diagrammes sont des éléments graphiques. Ceux-ci décrivent le contenu des vues, qui sont des notions abstraites. Les diagrammes peuvent faire partie de plusieurs vues.

VI.1. Vue fonctionnelle

La vue fonctionnelle cherche à appréhender les interactions entre les différents acteurs/utilisateurs et le système, sous forme dêobjectif à atteindre dûun côté et sous forme chronologique de scénarios dêinteraction typiques de lêautre.

Elle est représentée à lêaide des diagrammes comportementaux suivants:

VI.1.1 Diagramme de cas dêutilisation

Les cas dêutilisation sont utiles lors de lêélaboration du cahier des charges ou du document de spécifications des besoins du logiciel.

Un cas dêutilisation (use case) modélise une interaction entre le système informatique à développer et un utilisateur ou acteur interagissant avec le système. Plus précisément, un cas dêutilisation décrit une séquence dêactions réalisées par le système qui produit un résultat observable pour un acteur.

Les acteurs se représentent sous forme de petits personnages qui déclenchent les cas. Ces derniers se représentent par des ellipses contenues dans un rectangle représentant le système.

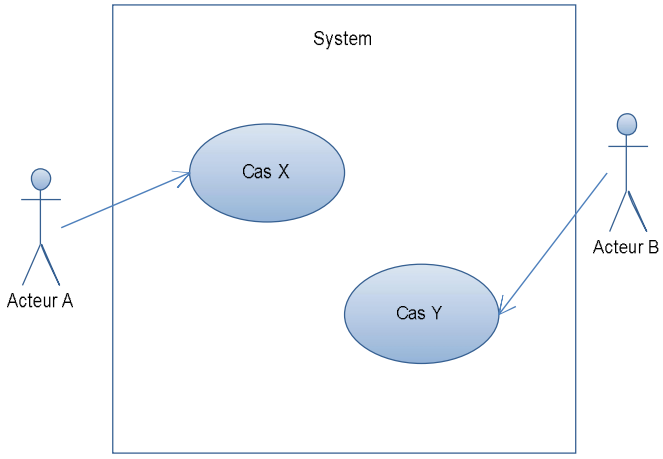

Figure 1: diagramme de cas d'utilisation.

Liens entre cas d'utilisation : include **et** extend

- **Include :** Ou bien INCLUSION, cette relation permet de décomposer des comportements et de définir des comportements partageables entre plusieurs cas d'utilisation. Alors elle indique que le comportement de cas d'utilisation destination est inclus dans le cas d'utilisation source.
- **Extend :** Ou bien EXTENSION, Celle-ci permet de modéliser les variantes de comportement d'un cas d'utilisation. Donc elle indique que le cas d'utilisation source ajoute son comportement au cas d'utilisation destination.

VI.1.2 Diagramme d'état-transition

Le diagramme d'état représente la façon dont évoluent, durant le processus les objets appartenant à une même classe. La modélisation du cycle de vie est essentielle pour représenter et mettre en forme la dynamique du système.

VI.1.3 Diagramme d'activité

Le diagramme d'activités représente les règles d'enchainement des activités dans le système. Il permet d'une part de consolider la spécification d'un cas d'utilisation, d'autre part de concevoir une méthode

VI.2. Vue structurelle

La vue structurelle, ou statique s'occupe de la structuration des données et tente d'identifier les objets/composants constituant le programme, leurs attributs, opérations et méthodes, ainsi que les liens ou associations qui les unissent.

Elle regroupe les diagrammes suivants :

VI.2.1 Diagramme de classes

Un diagramme des classes décrit le type des objets ou données du système ainsi que les différentes formes de relation statiques qui les relient entre eux. On distingue classiquement deux types principaux de relations entre objets :

- les associations, bien connues des vieux modèles entité/association utilisés dans la conception des bases de données depuis les années 70 ;
- les sous-types, particulièrement en vogue en conception orientée objets, puisqu'ils s'expriment très bien à l'aide de l'héritage en programmation.

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. Une classe permet de décrire un ensemble d'objets (attributs et comportement), tandis qu'une relation ou association permet de faire apparaître des liens entre ces objets. On peut donc dire :

- un objet est une instance de classe
- un lien est une instance de relation

L'intérêt de ce diagramme est la modélisation des entités de système d'information, autrement dit est un modèle permettant de décrire de manière abstraite et générale les liens entre objets.

VI.2.2 Diagramme d'objets

Les diagrammes d'objets servent à inventorier les objets (i.e. les instances de classes) composant une application à un instant donné ainsi que les relations et donnent une image statique des relations entre ces objets. Ils peuvent également être mis en î uvre pour tester la pertinence d'un diagramme de classe.

VI.2.3 Diagramme de composants

Les diagrammes de composants servent à représenter la configuration logicielle ainsi que les relations d'un système; on permettant également de représenter les programmes, les sous-programmes et les interrelations.

VI.2.4 Diagramme de déploiement

Le diagramme de déploiement correspond à la fois à la structure du réseau informatique qui prend en charge le système logiciel, et la façon dont les composants d'exploitation y sont installés. Ce diagramme a pour but d'indiquer la disposition et l'organisation des différents matériels (équipements) composant un système, ainsi que la disposition des programmes exécutables sur ceux-ci.

VI.2.5 Diagramme de paquetage

Un paquetage étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, le Diagramme de paquetage sert à représenter les dépendances entre paquetages, cêst-à-dire les dépendances entre ensembles de définitions.

VI.2.6 Diagramme de structure composite

Depuis UML 2.x, le diagramme de structure composite permet de décrire sous forme de boîte blanche les relations entre composants d'une classe.

VI.3. Vue dynamique

Cette vue est plus algorithmique et orientée « traitement », elle vise à décrire lêévolution (la dynamique) des objets complexes du programme tout au long de leur cycle de vie.
De leur naissance à leur mort, les objets voient leurs changements dêétats guidés par les interactions avec les autres objets.
Elle regroupe les diagrammes suivants :

VI.3.1 Diagramme de séquence

Les diagrammes de séquences mettent en valeur les échanges de messages (déclenchant des événements) entre acteurs et objets (ou entre objets et objets) de manière chronologique, lêévolution du temps se lisant de haut en bas.
Chaque colonne correspond à un objet (décrit dans le diagramme des classes), ou éventuellement à un acteur, introduit dans le diagramme des cas. La *ligne de vie* de lêobjet représente la durée de son interaction avec les autres objets du diagramme.

VI.3.2 Diagramme de communication

Il permet une représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.

VI.3.3 Diagramme d'interaction

Il permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité).

VI.3.4 Diagramme de temps

Il permet de décrire les variations d'une donnée au cours du temps.

VI. Modéliser avec UML

UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles. De ce fait, il faudra choisir un processus (démarche à suivre) pour pouvoir modéliser avec ce langage. Plusieurs processus existent parmi eux le Processus Unifier UP.

VII. Processus UP [5]

VII.1 Introduction

Le processus unifié UP (Unified Process) est une méthode de prise en charge du cycle de vie d'un logiciel orienté objet.

L'OMG (Object Management Group) qui standardise les technologies de l'objet a standardisé UML qui est utilisable par toutes les méthodes objet et lisible à la fois par les humains (dans sa forme graphique) et les machines avec sa syntaxe précise, UML n'impose pas de processus, cependant, il existe une démarche naturelle qui parle des cas d'utilisation et qui exprime un point de vue fonctionnel sur le système. Cette démarche est la suivante :

- Cas d'utilisation : spécification initiale du système
- Diagrammes de séquences : associés aux cas d'utilisation
- Diagrammes de classes

VII.2 Les phases dôUP

VII.1.1 Initialisation

Définir la portée et la faisabilité du projet afin de décider de sa poursuite ou son arrêt.

VII.1.2 Elaboration

Poursuit trois objectifs principaux en parallèle

- Identifier et décrire la majeure partie des besoins des utilisateurs
- Construire lôarchitecture de base du système
- Lever les risques majeurs du projet

VII.1.3 Construction

Consiste surtout à concevoir et implémenter lôensemble des éléments opérationnels.

VII.1.4 Transition

- Consiste surtout à concevoir et implémenter lôensemble des éléments opérationnels.
- Faire passer le système informatique des mains des développeurs aux mains des utilisateurs.

Chaque phase est elle-même décomposée séquentiellement en itérations limités dans le temps.

Les activités du développement sont définies par cinq disciplines fondamentales, qui sont :

- ✓ La capture des exigences
- ✓ Lôanalyse et la conception
- ✓ Lôimplémentation
- ✓ Le test
- ✓ Le déploiement

VIII. Conclusion

Comme UML n'impose pas de méthode de travail particulière, il peut être intégré à n'importe quel processus de développement logiciel de manière transparente. Parmi les processus cités précédemment, nous avons choisi le processus UP quôon utilisera pour modéliser notre système. Cette modélisation sera détaillée dans le chapitre suivant.

Chapitre 2 : Modélisation du système GDFE

I. Introduction

Ce chapitre porte sur la conception du système comme nous avons déjà cité dans le chapitre précédent, notre choix s'est porté sur le processus UP, en conséquence, nous allons détailler trois étapes:

- Tout d'abord, nous commencerons par spécifier les besoins de notre système en définissant le diagramme de cas d'utilisation.

- Ensuite les diagrammes de cas d'utilisation vont être détaillés en plusieurs diagrammes de séquences.

- Nous terminerons par représenter le diagramme de classes qui décrit la structure statique de notre système.

II. Définition du système GDFE

Le système de gestion de demande de formation dans une entreprise gère en interne un catalogue de formations classées par thèmes. Le catalogue propose un descriptif pour chaque formation et offre la possibilité de s'inscrire à une ou plusieurs formations qui vont être suivies par un formateur spécialisé.

- **Spécification des besoins**
  - Les employés de l'entreprise peuvent visualiser le catalogue des formations et faire une demande d'inscription.
  - La demande d'inscription doit d'abord être validée par son supérieur hiérarchique, toujours via l'application, avant d'être comptabilisée.
  - Un employé peut suivre ses demandes.
  - Quand la demande est soit validée soit refusée, l'employé reçoit un email avec la raison du refus dans le cas du refus.

III. Modélisation du système GDFE

Pour modéliser le système GDFE, on commence par partager ce système en plusieurs cas d'utilisations. Chaque cas d'utilisation correspond à un ou plusieurs scénarios.

III.1 Diagramme de cas dôutilisation

Les cas dôutilisations et les acteurs GDFE sont schématisés dans le diagramme représenté à la Figure 2. Le diagramme montre aussi lôinteraction entre chaque cas dôutilisation et les acteurs du système.

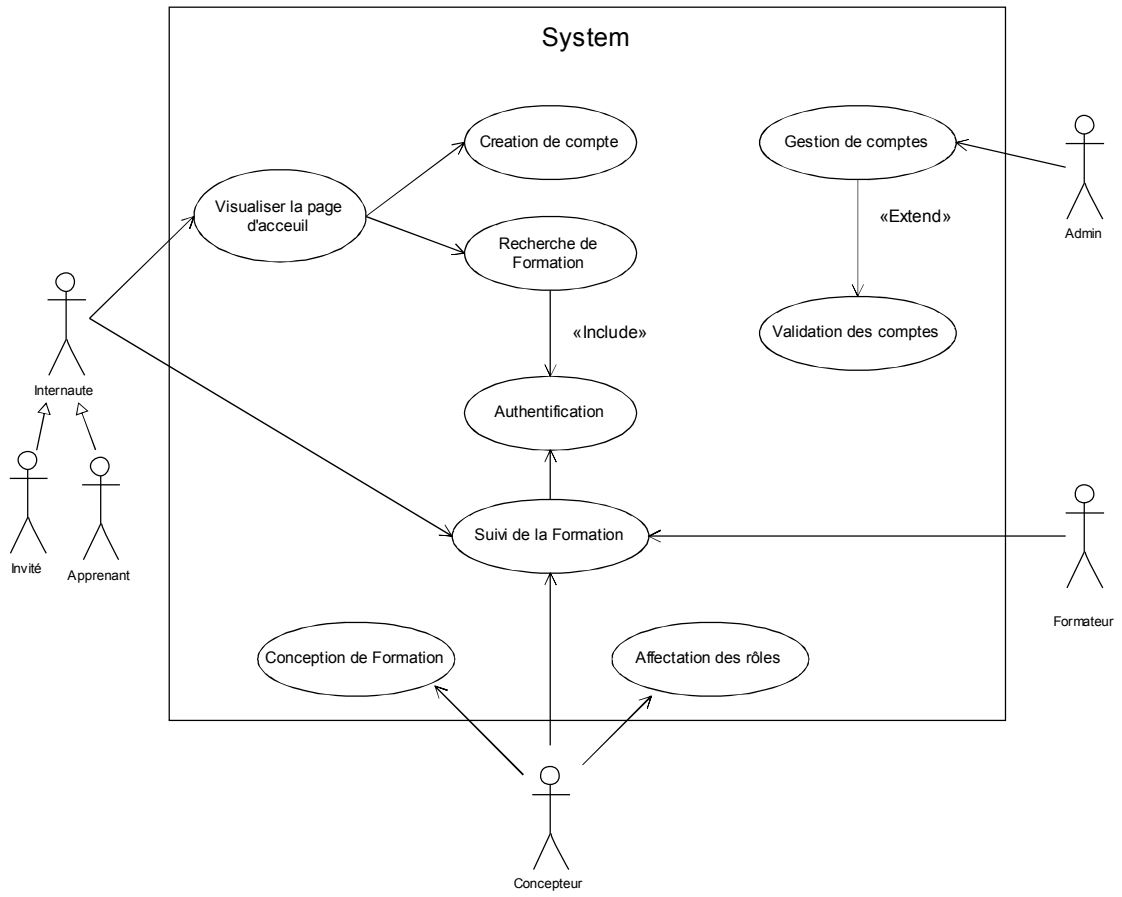

Figure 2: diagramme de cas dôutilisation « gestion de demande de formation »

III.1.1 Identification des acteurs GDFE

Chaque acteur impliqué dans le système a un rôle. Cet acteur peut être une personne ou un groupe de personne. Les acteurs interagissant avec le système GDFE sont :

- Lôinternaute (qui peut être un apprenant ou un invité)
- Lôadministrateur
- Le concepteur des formations
- Le formateur (responsable du déroulement de la formation)

Chaque scénario sera détaillé dans ce qui suit.

III.2 Les diagrammes de séquence

III.2.1 Visite du catalogue

Un internaute se connecte au site, il peut visualiser le catalogue et quitter.

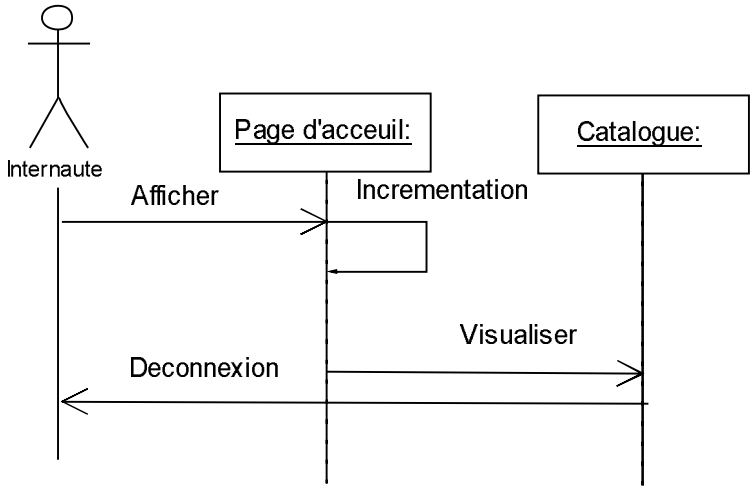

Figure 3: diagramme de séquence « visite du catalogue ».

III.2.2 Inscription (choix de la formation inclus)

Lorsque lôinternaute visualise le catalogue et désire sôinscrire dans une ou plusieurs formations il doit dôabord sôauthentifier.

- Si le compte existe, lôinternaute (et dans ce cas là, côest un apprenant) accède à la liste des formations et fait son inscription.
- Sinon, message dôerreur « compte invalide »

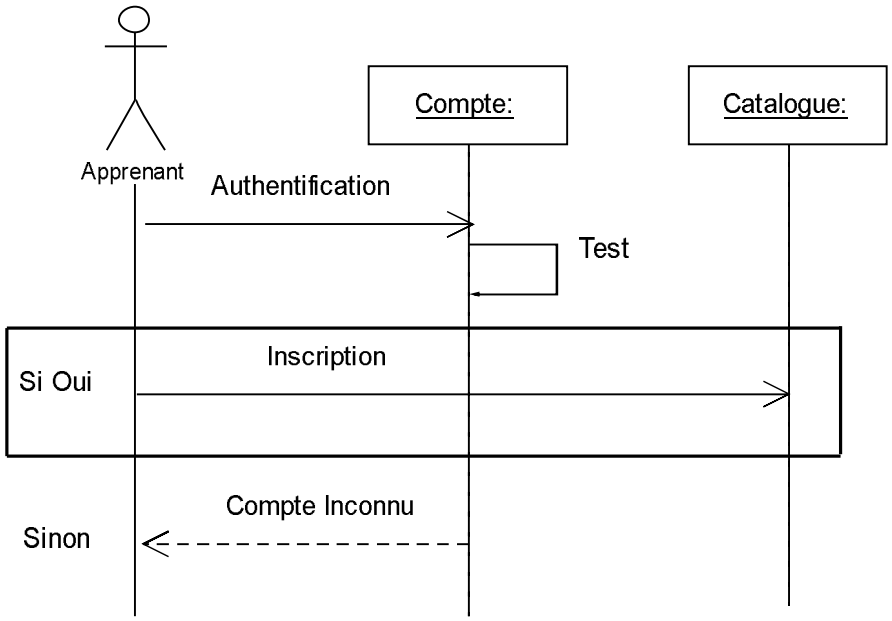

Figure 4: diagramme de séquence « Inscription ».

III.2.3 Création de compte

Lôinscription dans une formation nécessite la possession dôun compte utilisateur

- Si le compte nôexiste pas lôapprenant doit alors en créer un.
- Lôadministrateur valide toujours la création.

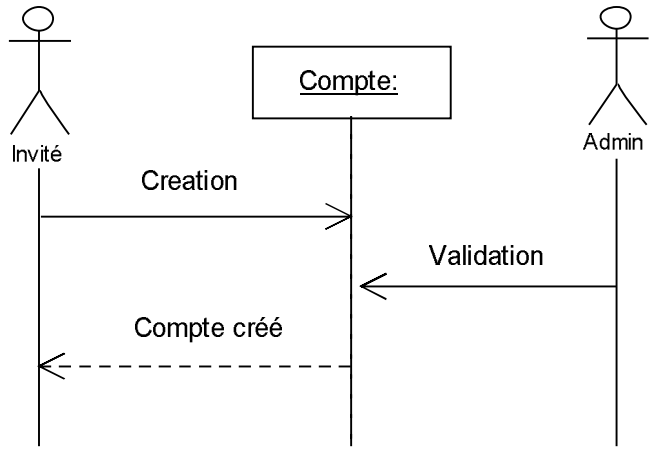

Figure 5: diagramme de séquence « Création de compte ».

III.2.4 Validation ou annulation de l'inscription

Une fois la formation choisie et l'inscription faite, c'est le tour de l'administrateur de la validée ou l'annulée.

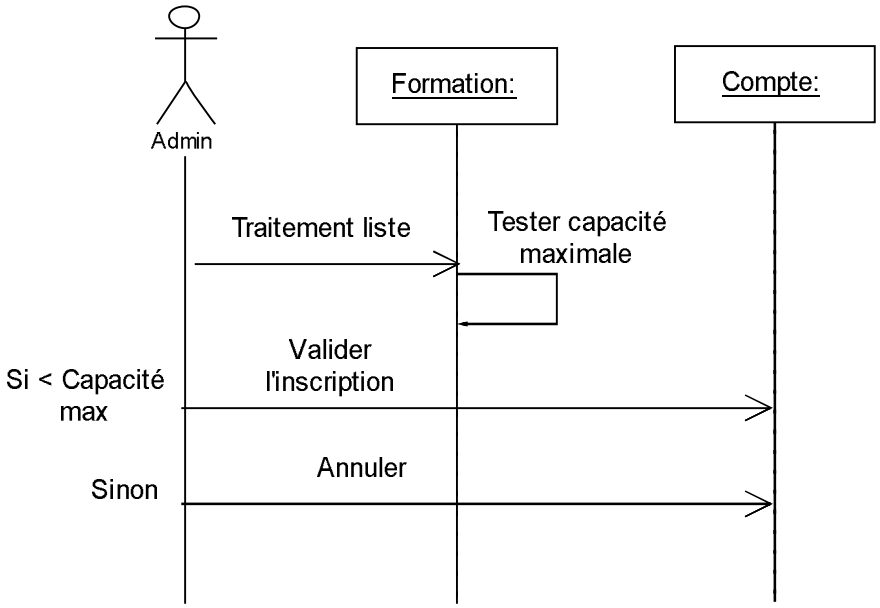

Figure 6: diagramme de séquence « Validation ou annulation de l'inscription ».

III.2.5 Mise en ligne d'une formation

La formation passe par trois phases pour être accessible :

- Création (titre, description, nombre de sessions...)-
- Conception(les cours...) : Tant qu'elle n'est pas achevée, la mise en ligne n'aura pas lieu.
- Publication ou mise en ligne

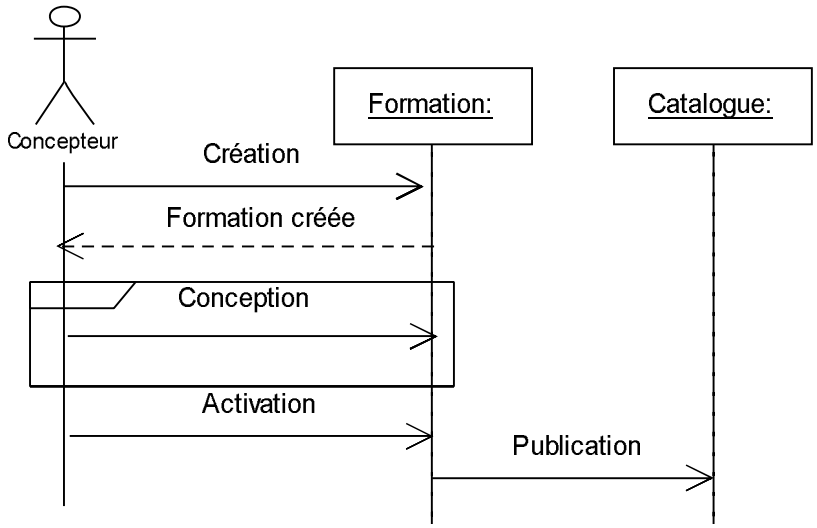

Figure 7: diagramme de séquence « Mise en ligne dôune formation

III.2.6 Affectation des rôles

Pour quôun concepteur puisse affecter une tache à un formateur il doit dôabord vérifier les compétences de ce dernier. Sôil ne répond pas au besoin, la formation reste inactive.

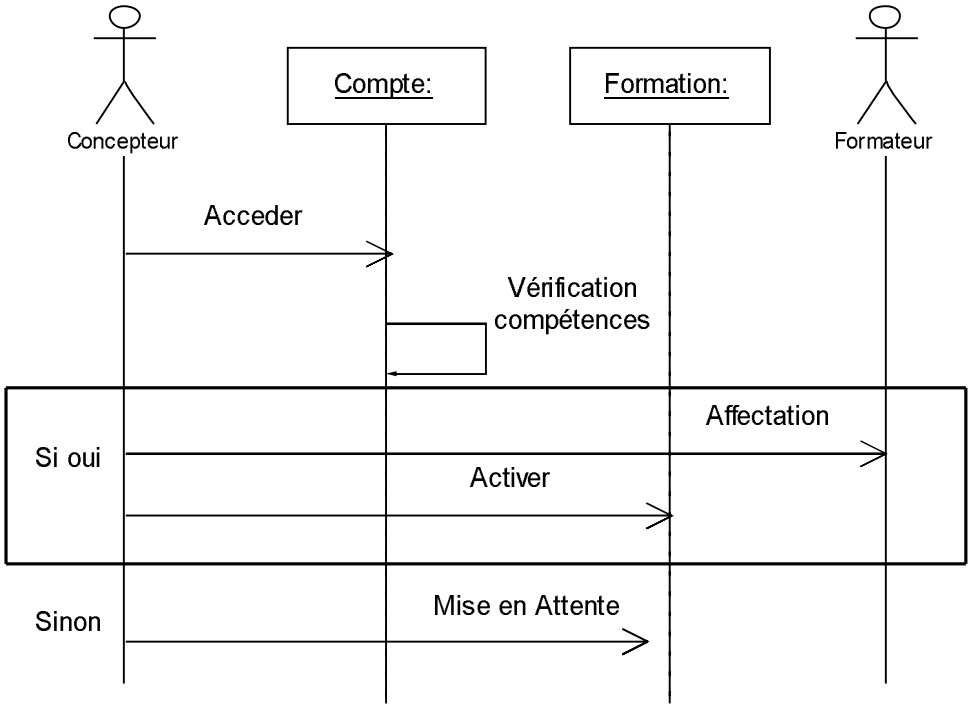

Figure 8: diagramme de séquence « Affectation des rôles ».

III.2 Diagramme de classe

Le diagramme de classes identifie les classes de notre système et les associations entre elles. Le diagramme contient dix classes.

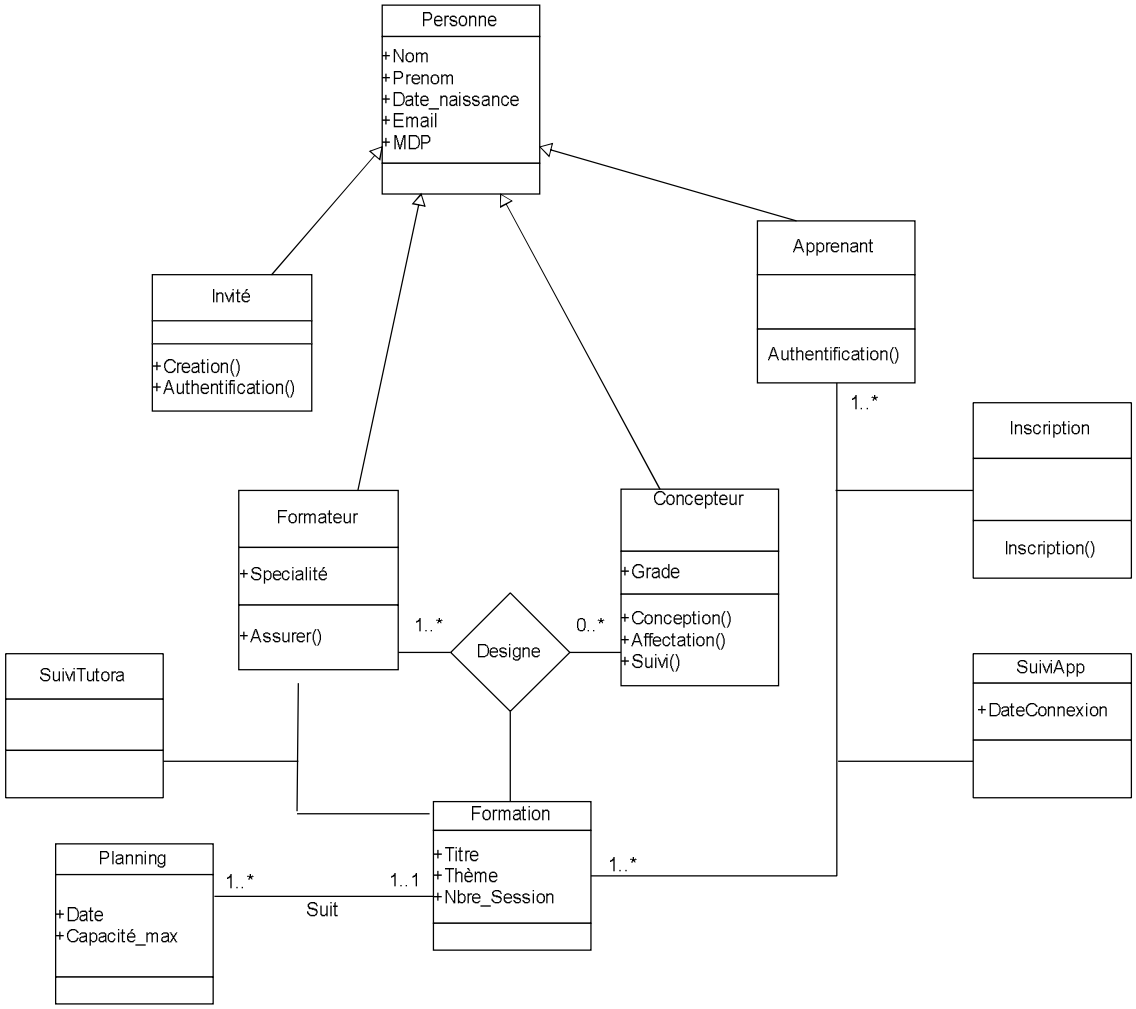

Figure 9: diagramme de classes

IV. Modèle logique de données GDFE

Le modèle logique présenté ci-dessous a été obtenu en appliquant les règles de passage [6] sur le diagramme de classes (Figure 9).

Apprenant (idApprenant, nom, prenom, date_naiss, e-mail, mdp)

Concepteur (idConcepteur, nom, domaine)

Formateur (idFormateur, nom_formateur, compétence)

Formation (idFormation, idFormateur*, titre, theme)

Inscription (idInscription, idApprenant*, idFormation*, date_inscrip)

Apprentissage (idApprentissage, idFormation*, idApprenant*, date_connexion)

Désignation (idDesignation, idConcepteur*, idFormateur*, date_design)

Planning (idPlanning, idFormation*, date, capacité_max)

V. Conclusion

Nous avons réalisé dans ce chapitre la conception UML du système GDFE. Cette conception est une étape nécessaire et très importante pour pouvoir créer notre base de données et réaliser lôapplication, le prochain chapitre détaillera les étapes de lôimplémentation de lôapplication GDFE.

Chapitre 3 : Implémentation du Système GDFE

I. Introduction

Dans ce chapitre, nous allons justifier le choix du langage, donner un bref aperçu sur les outils utilisés, présenter les résultats de notre travail, et finir par une conclusion et quelques perspectives.

II. Choix du langage de programmation

Notre choix de langage c'est rapidement porté sur java qui est un langage orienté objet très utilisé, notamment par un grand nombre de programmeurs professionnels ce qui en fait un langage de haut niveau. Java intègre les concepts les plus intéressants des technologies informatiques récentes dans une plate-forme de développement riche et homogène.

L'approche objet de ce langage, mais aussi sa portabilité et sa gratuité, en font un des outils de programmation idéaux pour s'initier à la programmation objet.

II.1 Définition JAVA [7]

Java est un langage objet permettant le développement d'applications complètes s'appuyant sur les structures de données classiques (tableaux, fichiers) et utilisant abondamment l'allocation dynamique de mémoire pour créer des objets en mémoire.

La notion de structure, ensemble de donnée décrivant une entité (un objet en Java) est remplacée par la notion de classe au sens de la programmation objet. Le langage Java permet également la définition d'interfaces graphiques (GUI : Graphical User Interface) facilitant le développement d'applications interactives et permettant à l'utilisateur de « piloter » son programme dans un ordre non imposé par le logiciel.

II.2 Principales caractéristiques de JAVA

Un programme Java est portable au sens où il peut s'exécuter sur des ordinateurs fonctionnant avec différents systèmes d'exploitation. Les programmes écrits en Pascal ou en langage C sont aussi portable par compilation du code source sur la machine où le programme doit s'exécuter. Java est portable d'une plate-forme à une autre sans recompilation. Le compilateur produit un langage intermédiaire appelé « bytecode » qui est interprété sur les différentes machines.

Il suffit donc de communiquer le bytecode et de disposer dôun interpréteur de bytecode pour obtenir lœexécution dôun programme Java.

On peut faire de nombreuses sortes de programmes avec Java :

- Applications, sous forme de fenêtre ou de console
- Applets Java, qui sont des programmes Java incorporés à des pages web
- Applications pour appareils mobiles, avec J2ME é

III. Choix de lœnvironnement de développement (Netbeans)

En ce qui concerne le développement de lôapplication, nous avons choisi NetBeans qui est un environnement de développement en java open source. Le produit est composé d'une partie centrale à laquelle il est possible d'ajouter des modules supplémentaires (pour la création de diagramme UML, pour la génération de code en dœautres langages, é).

En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, 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, SPARC), Mac OSX et Open VMS.

IV. Choix du SGBD

NetBeans comprend un explorateur de bases de données qui supporte toutes les bases relationnelles pour lesquelles un connecteur JDBC existe (selon les versions des gestionnaires de bases de données): JavaDB (Derby) MySQL, PostgreSQL, Oracle, Microsoft SQL, Interbase (Delphi)é

L'explorateur comprend un éditeur de requêtes, un gestionnaire intégré de bases de données MySQL.

Nous avons choisi MySQL comme SGBD car cœst un SGBDR, mais aussi pour sa puissance et sa forte appréciation par les développeurs.

V. Définition de MySQL

MySQL est un système de gestion de base de données (SGBD). Selon le type d'application, sa licence est libre ou propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle et Microsoft SQL Server.

V.1. Principales caractéristiques de MySQL

MySQL est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multi-thread et multi-utilisateur.

VI. Modèles MVC [8]

Le Modèle-Vue-Contrôleur (en abrégé MVC, de l'anglais *Model-View-Controller*) est une architecture et une méthode de conception qui organise l'interface homme-machine (IHM) d'une application logicielle. Ce paradigme divise l'IHM en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle précis dans l'interface.

Ce modèle d'architecture impose la séparation entre les données, la présentation et les traitements, ce qui donne trois parties fondamentales dans l'application finale : le modèle, la vue et le contrôleur.

VI.1. Model

Le modèle représente le comportement de l'application : traitements des données, interactions avec la base de données, etc. Il décrit ou contient les données manipulées par l'application. Il assure la gestion de ces données et garantit leur intégrité. Dans le cas typique d'une base de données, c'est le modèle qui la contient. Le modèle offre des méthodes pour mettre à jour ces données (insertion, suppression, changement de valeur). Il offre aussi des méthodes pour récupérer ces données. Les résultats renvoyés par le modèle sont dénués de toute présentation.

VI.2. Vue

La vue correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, é). Ces différents événements sont envoyés au contrôleur. La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle et d'interagir avec l'utilisateur.

VI.3. Contrôleur

Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle, ce dernier avertit la vue que les données ont changée pour qu'elle se mette à jour. Certains événements de l'utilisateur ne concernent pas les données mais la vue. Dans ce cas, le contrôleur demande à la vue de se modifier. Le contrôleur n'effectue aucun traitement, ne modifie aucune donnée. Il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondant à la demande.

VII. Description de lôapplication

Nous allons maintenant présenter les principaux écrans de lôapplication.

- **Ecran dôaccueil**

  Côest la page dôaccueil et à ce niveau là, lôapprenant doit sôauthentifier afin dôaccéder au catalogue. Si les informations sont incorrectes, le message (óveuillez vérifier votre mot de passeô) sera affiché. (Figure 10)

Figure 10: Ecran dôauthentification.

- **Ecran du catalogue**

Dans cette page, lôapprenant trouve les formations sous forme dôarborescence et il choisi la formation selon le thème souhaité.

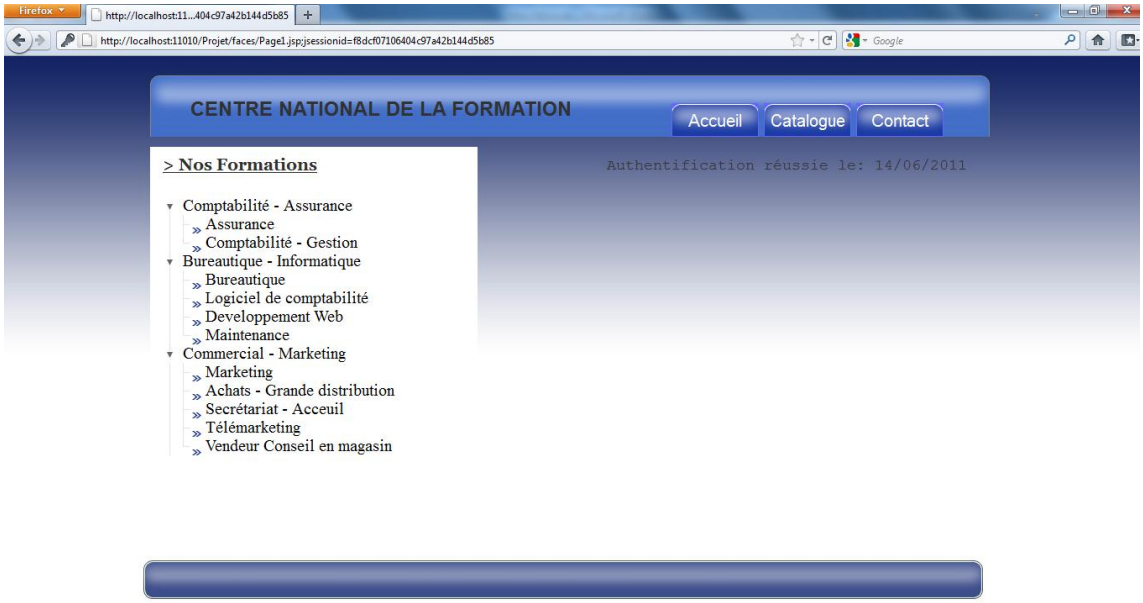

Figure 11: Ecran du catalogue

- **Ecran inscription**

Si la personne qui visite la catalogue est invité (c'est-à-dire ne possédant pas de compte dôutilisateur) et il désire sôinscrire, en cliquant sur le lien sôinscrire, il trouvera un formulaire à remplir dans la page dôinscription. (Figure 12)

Figure 12: Ecran dôinscription

- **Ecran choix de formation**

Lorsquôun apprenant choisi une formation, une page sôaffiche en présentant quelque objectif de cette formation et offrant la possibilité de sôinscrire une et une seule fois.

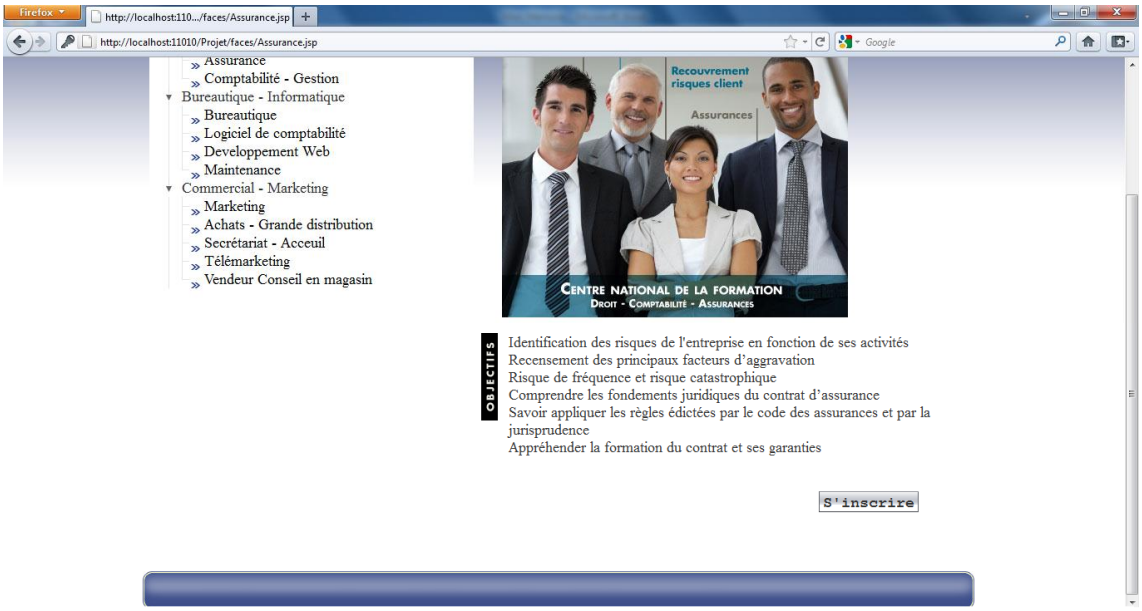

Figure 13: Ecran détail formation

Si lôinscription sôeffectue sans aucun problème le message ó Inscription effectuée avec succèsô sôaffichera (figure 14), sinon lôapprenant sera renvoyé à la page du catalogue pour faire un autre choix ou quitter si désiré.

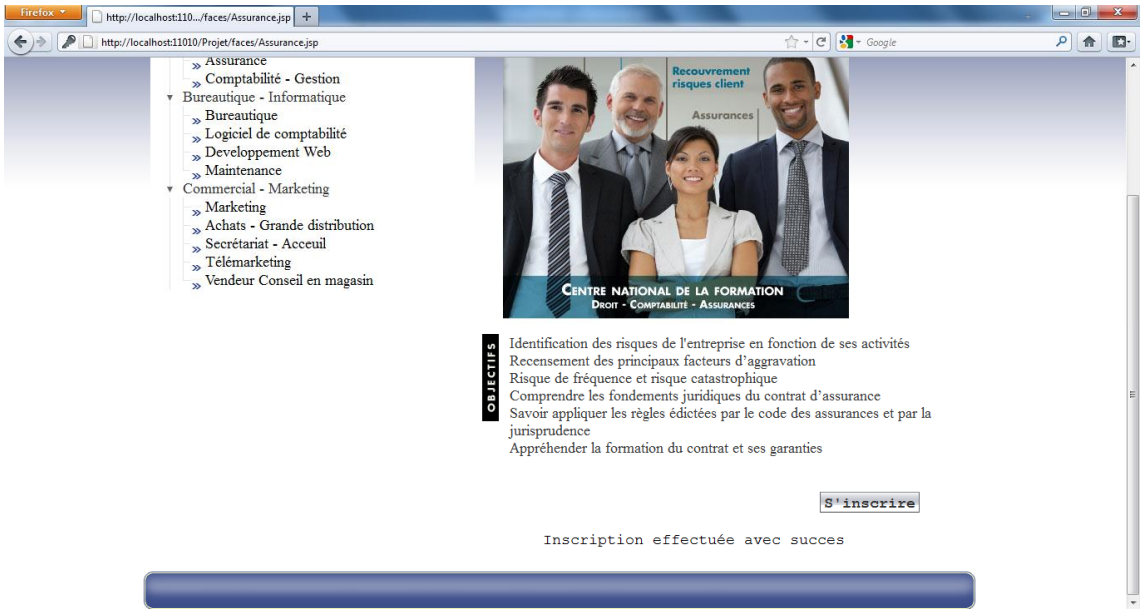

Figure 14: Ecran inscription dans une formation

VIII. Conclusion

Dans ce dernier chapitre, nous avons présentés le coté réalisation de notre projet, justifier le choix du langage, lôoutil de développement, nous avons parlés aussi des MVC et enfin nous avons finis par présenter les interfaces les plus importantes.

Conclusion Générale

Ce projet a contribué à améliorer mes connaissances dans plusieurs domaines. Il m'a permis d'améliorer mes connaissances en conception objet et d'apprendre un nouveau langage de programmation

Nous avons appliqué au maximum possible les recommandations de conduite dans un projet de gestion afin d'avoir une application qui soit performante. J'ai utilisé UML pour modéliser le système et le langage Java pour implémenter l'application.

Pour la mise en place de notre système, notre choix s'est porté sur une application web avec une architecture trois tiers supportée par le modèle JSF

Les futures modifications prévues pour cette application :

- Ajout de fonctionnalités administratives afin de gérer la liste des apprenants.
- Introduire la technologie Ajax pour une vérification simultanée de la disponibilité des champs entrés par un invité désirant créer un compte.
- Enrichir le catalogue avec d'autres formations

Références bibliographiques

[1] Olivier Sigaud, Introduction à la modélisation orientée objets avec UML, Edition 2005-2006
[2] J.STEFFE, COURS UML13.doc, ENITA de Bordeaux, janvier 2003

[3]Yohann RICHARD, Modélisation avec UML, Février 2000

[4] http://fr.wikipedia.org/wiki/Unified_Modeling_Language

[5] MR CHOUITI, UP et les méthodes agiles, Master 1 SIC Université Abou Bakr Belkaidï Tlemcen, 2010-2011.

[6] Gilles Roy, Conception de bases de données avec UML, Presses de lôUniversité du Québec, 2009

[7] Emmanuel Puy baret, Les cahiers du programmeur Java 1.4 et 5.0, Edition Eyrolles 2006.
[8] http://fr.wikipedia.org/wiki/Modè-Vue-Contrôleur