VIETNAM NATIONAL UNIVERSITY (VNU)
INSTITUT FRANCOPHONE INTERNATIONAL (IFI)

Option : Systèmes Intelligents et Multimédia (SIM)

Promotion : XXI

GENIE LOGICIEL AVANCE (GLA)

« Modélisation et conception d'un gestionnaire de tâche avec le langage de programmation oriente objet JAVA »

Rapport du Projet

RAKOTOARIVELO Anjara Nobby

Encadrant :

Monsieur HO Tuong Vinh (VNU, IFI)

année académique 2016-2017

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

1 Introduction

1.1 Contexte et objectif

Le but de ce travail étant de faire une remise à niveau des étudiants de l'IFI dans la Programmation Orienté Objet JAVA. C'est dans ce contexte que se présente notre projet actuel qui consiste à concevoir et développer une application de gestionnaire des tâches. Généralement, les exigences du projet sont d'abord : l'ajout, la suppression et la mise à jour des tâches et des membres. Ensuite, assigner un tâches à un membre. Enfin, faire des recherche selon le statu des tâches ou les tâches effectue par un membre donne.

1.2 Plan de travail

Généralement, le projet est subdivise en 3 parties : d'abord, la présentation des exigences des utilisateurs. Ensuite, la phase de conception et du présentation de l'application. Enfin, la phase de test et du déploiement.

2 Présentation du travail

2.1 Spécification des exigences

C'est une étape primordiale du projet et le point de départ de tous projet informatique. EN effet, elle permet de déterminer les exigences fonctionnelles ou non fonctionnelles (norme ISO 9126) de l'application. En d'autres termes, c'est un compte rendu entre le Client et les Utilisateurs finaux avec l'équipe de développement.

En effet, construire un diagramme des cas d'utilisation est la meilleur manières d'exprimer les exigences d'une application.

2.1.1 Diagramme des cas d'utilisations

Le diagramme des cas d'utilisation permet de déterminer les exigences des utilisateurs pour une nouvelle application. En d'autres termes, les cas d'utilisation énumèrent toutes les fonctionnalités possible du future application. L'ensemble des cas d'utilisation spécifie la fonctionnalité complète du système. Notons cependant que le diagramme de cas d'utilisation montre un système uniquement au point de vue des acteurs [Ledang 2001 ; Audibert 2009]. Notre Diagramme des cas d'utilisation est représentée par la (FIGURE 1).

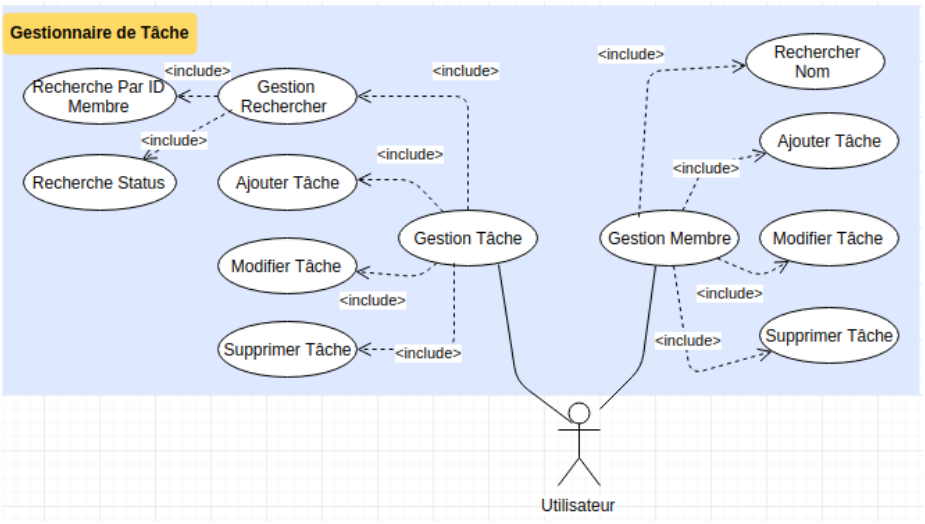

FIGURE 1 – Diagramme des cas d'utilisation

2.1.2 Description des cas d'utilisations

Le (TABLEAU 1) nous donne les détails de chaque cas d'utilisation de notre diagramme des cas d'utilisation (FIGURE 1).

Numéro	Nomination	Description
1	Gestion Tâche	permet à l'utilisateur d'accéder au menu tâche et ainsi de choisir l'action effectué(ajouter, modifier, supprimer ou rechercher).
2	Ajouter Tâche	permet à l'utilisateur d'ajouter une nouvelle tâche dans la liste.
3	Supprimer Tâche	permet à l'utilisateur de supprimer un instance de tâche dans la liste.
4	Modifier Tâche	permet à l'utilisateur de modifier une instance de tâche dans la liste.
5	Gestion de Recherche	permet à l'utilisateur de choisir la recherche effectuée (Recherche Par ID Membre ou Recherche Par STATUS).
6	Recherche Par ID Membre	permet à l'utilisateur de recherche une tâche par rapport à l'identifiant du membre qui effectue la tâche.
7	Recherche Par STATUS	permet à l'utilisateur de recherche une tâche par rapport à son STATUS(nouveau, en progrès ou terminé).
8	Gestion Membre	permet à l'utilisateur d'accéder au menu membre et ainsi de choisir l'action effectué(ajouter, modifier, supprimer ou rechercher).
9	Ajouter Membre	permet à l'utilisateur d'ajouter un nouveau membre dans la liste.
10	Supprimer Membre	permet à l'utilisateur de supprimer une instance de membre dans la liste.
11	Modifier Membre	permet à l'utilisateur de modifier un instance de membre dans la liste.

TABLE 1 – Description des cas utilisations

2.2 Conception

Cette partie consiste a déterminer l'environnement technique adéquat pour déterminer la manière de résoudre le problème posé. Pour arriver à résoudre le problème, quelques étapes sont nécessaires représentés par la (FIGURE 2). [CHARROUX 2008]

Cependant, pour effectuer ses étapes, un langage de modélisation est essentiel avec plusieurs caractéristiques :

- Générique ;
- Expressif ;
- Flexible (configurable, extensible) ;
- Syntaxe et sémantique.

Ses divers caractéristiques sont présent dans le langage de modélisation objet UML. Se qui nous a conduit de choisir Unified Modeling Language (UML) pour la conception et modélisation de notre système. [MULLER et GAERTNER 2000]

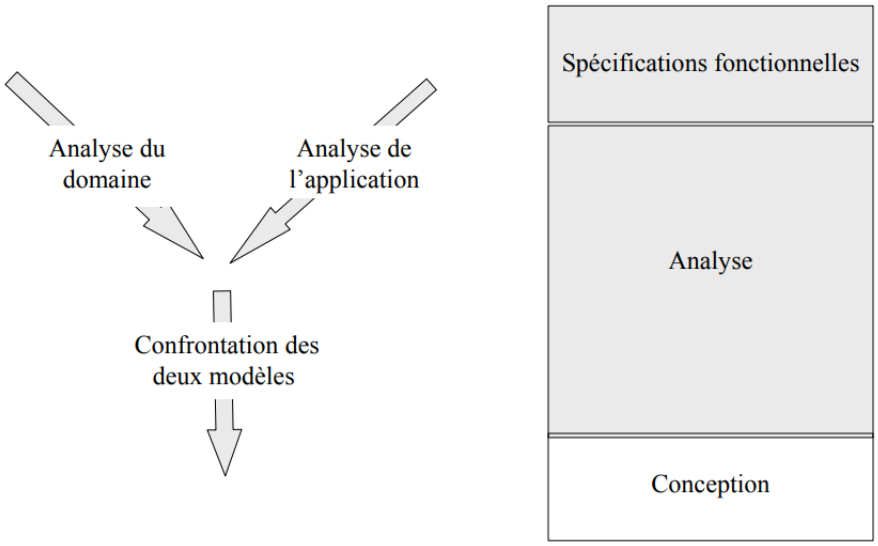

FIGURE 2 – Méthode de modélisation objet

2.2.1 Diagramme des classes

Le diagramme de classes permet de mettre en évidence la structure interne de l'application. En d'autre termes, une représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation déjà définis dans la partie précédant. Cependant, Il s'agit d'une vue statique, car les facteur temporel ne sont pas pris en compte. Notre Diagramme des cas d'utilisation est représentée par la (FIGURE 3). [AUDIBERT 2009]

FIGURE 3 – Diagramme de classe

Description diagramme des classes :

membre 1 et tâche 1..* : décrit que chaque membre peut être assignes à un ou plusieurs tâches. Par contre chaque tâche ne peut être effectué que par un membre.

2.2.2 Diagramme de séquence

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation UML. Ses diagrammes permettent de visualiser les scénarios présentés pour chaque cas d'utilisation de notre diagramme des cas d'utilisations.

Normalement, toutes les méthodes décrient dans le diagramme de classe devraient être présentés dans notre diagrammes de séquences. Dans notre rapport, nous allons prendre quelques exemples(ajout tâche et modifier tâche).

Diagramme de séquence pour le cas ajouter tâche : La (FIGURE 4) nous présente le diagramme de séquence pour le cas d'un ajout d'une nouvelle tâches.

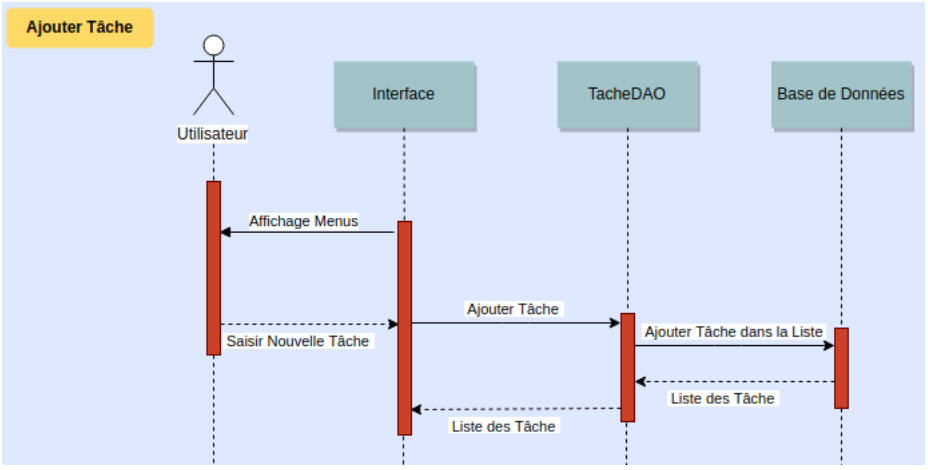

FIGURE 4 – Diagramme de séquence pour l'ajout des tâches

Description du diagramme de séquence pour le cas ajouter tâche : Nous pouvons constater qu'au démarrage de l'application, le système affiche le menu. Ensuite, l'utilisateur saisi les informations et envoye au système. Le système envoye la requette au DAO (Data Access Object) pour pouvoir ajouter dans la liste. Enfin le système retourne une liste mis à jour à l'écran.

Diagramme de séquence pour le cas modifier tâche : La (FIGURE 5) nous présente le diagramme de séquence pour le cas d'une modification d'une nouvelle tâches entre l'utilisateur et et le système.

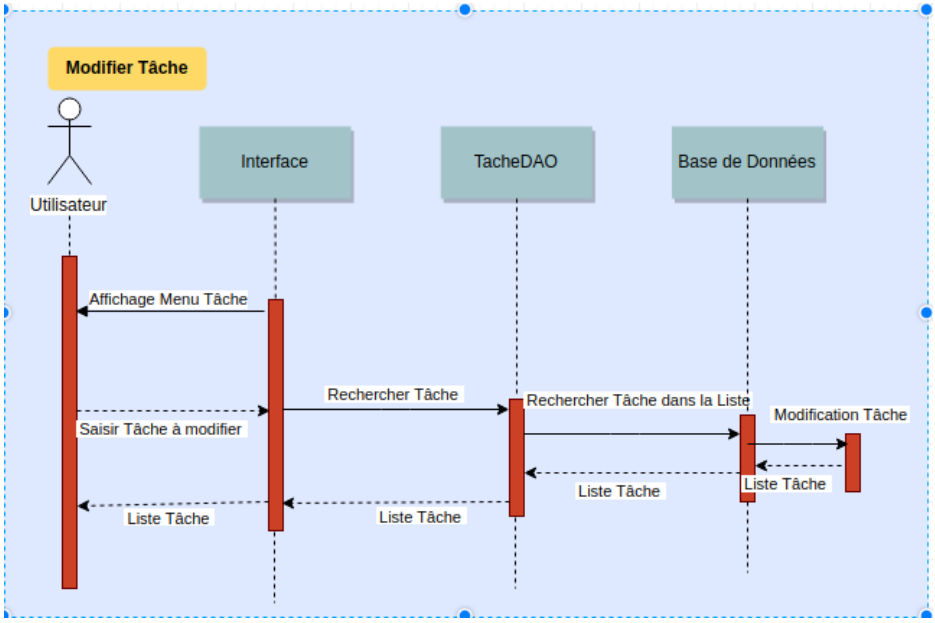

FIGURE 5 – Diagramme de séquence pour la modification des tâches

Description du diagramme de séquence pour le cas modifier tâche : Nous pouvons constater qu'au démarrage de l'application, le système affiche le menu. Ensuite, l'utilisateur saisi l'identifiant et les nouvelles informations du tâche à modifier et envoye au système. Le système envoye la requête au DAO (Data Access Object) pour récupérer dans la liste l'identifiant du tâche. A partir de l'identifiant, le DAO effectue une recherche de la liste des tâche à partir de son identifiant. S'il trouve l'identifiant, il modifie l'information et retourne la liste modifier à l'écran.

2.3 Implémentation

Dans cette partie du rapport, nous allons vous présenter l'architecture adopter ainsi que les outils utilises durant la phase de développement de notre application.

2.3.1 Architecture du modèle (MVC) avec 3-tiers

L'architecture MVC (Modèle Vue Contrôleur) a été adopter pour concevoir notre application. En résumé, cette architecture ne force pas l'utilisation d'un noyau regroupant toute l'application, et permet la répartition du code métier. Ce qui permet par la suite la réutilisabilité des codes. En effet, la couche modèle permet de décrire la zone. Ensuite, le contrôleur, qui joue le rôle d'intermédiaire (orchestration) entre les vues, le modèle et le couche d'accès a la base de données. Enfin, la dernière couche les vues, responsable d'affichage du programme. La (FIGURE 6) illustre la fonctionnement globale de l'architecture. [MARTIN, DEGRANDE et CHAILLOU 2005]

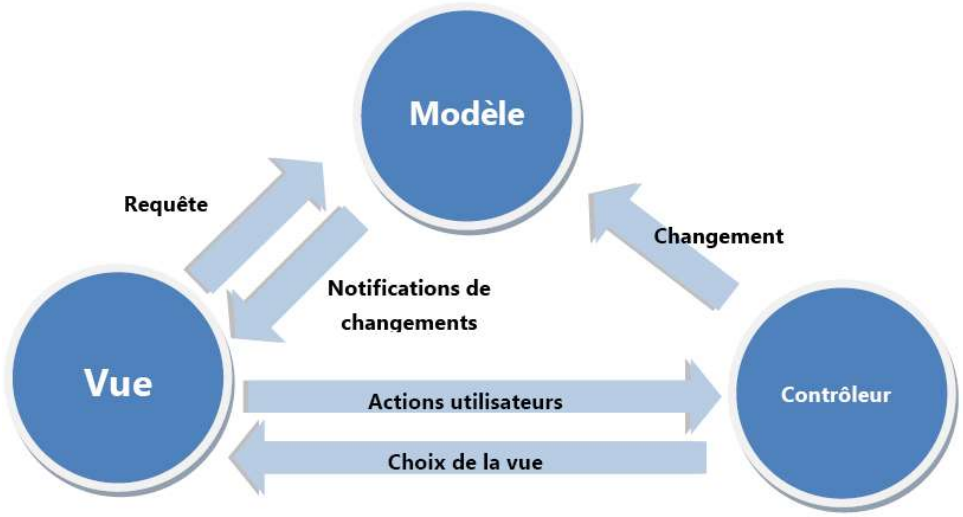

FIGURE 6 – Modèle Vue Contrôleur

Pour le cas de notre projet, nous avons 2 classes et qui va nous permettre d'avoir deux modèles (Membre et Taches). 1 contrôleur qui est en charge de l'orchestration et de l'affichage de l'application.

En outre, nous avons aussi opter pour l'utilisation de l'architecture 3-tiers pour accéder facilement aux données. Pour cela, nous avons utilise une 4^{e} couche la couche DAO(Data Access Object) (FIGURE 7) [MARSTON 2004]

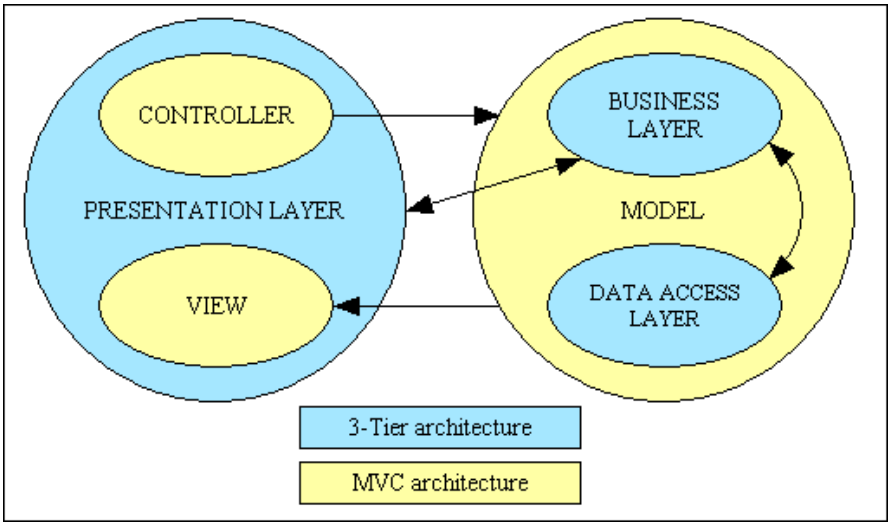

FIGURE 7 – Architecture 3 tiers

2.3.2 Langage de programmation (JAVA)

Le langage de programmation JAVA a été choisi pour implémenter notre application. Plusieurs raisons nous ont orienté à se choix.Tous d'abord, proposer

par notre professeur. Mais aussi, JAVA est un langage de programmation objet très populaire ce qui permet de chercher facilement des documentations et qui possède une forte communauté (forum) qui peut bien nous aider en cas de besoins. Enfin, JAVA est très bien adapter avec UML et la Programmation Orienté Objet (POO).

2.3.3 Base de Données (LIST)

En ce qui concerne les données, nous avons n'avons pas choisis l'utilisation d'une base de données conventionnel mais nous avons plutôt opté pour l'utilisation d'une liste (arraylist) pour sauvegarder notre données. Le choix se base sur la facilité et rapidité de déploiement de l'application. En outre, les données sont donc stocker dans la mémoire et s'initialise à chaque lancement de l'application ou démarrage de l'ordinateur.

2.3.4 Le choix des outils utilisés

- Ubuntu 16.04 comme système d'exploitation ;
- Github pour la gestion des versions et la collaboration ;
- IDE Netbean 8.1 pour le développement ;
- app.genmymodel.com pour la conception (UML)
- Junit pour les tests unitaires.

2.4 Présentation de l'application et manuel d'utilisation

Au démarrage de notre application, le menu principale (FIGURE 8) s'affiche d'abord. Alors, c'est à l'utilisateur de choisir s'il aller dans le menu tâche (saisir 1 dans la console) ou menu membre (saisir 2 dans la console) ou aussi quitter l'application (saisir 0 dans la console).

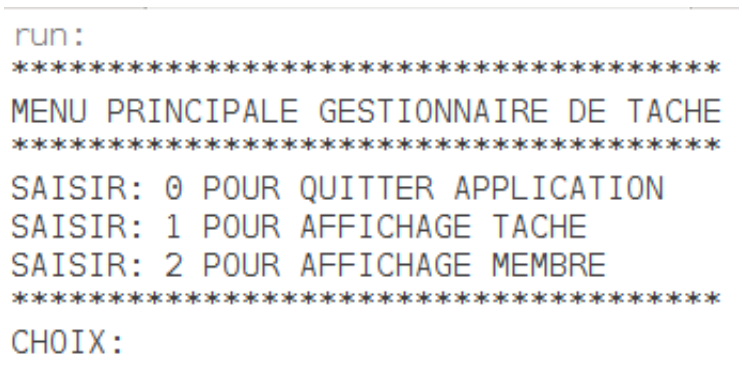

FIGURE 8 – Affichage du menu principale

Pour le cas de la tâche : si l'utilisateur a choisi d'aller dans le menu de mise à jour des tâche donc on obtient le résultat suivant (FIGURE 9). L'utilisateur dispose encore de plusieurs choix pour naviguer dans l'application (5 choix).

```
**************************************
MENU DE MISE A JOUR DE TACHE
**************************************
SAISIR: 0 POUR QUITTER APPLICATION
SAISIR: 1 POUR ASSIGNER TACHE
SAISIR: 2 POUR MODIFIER TACHE
SAISIR: 3 POUR SUPPRIMER TACHE
SAISIR: 4 POUR RECHERCHE TACHE
**************************************
CHOIX:
```

FIGURE 9 – Affichage du menu principale des taches

Arriver à cette étape, le menu assignation des tâches (FIGURE 10) permet l'assignation des tâche à un membre. Ce qui implique donc que le membre doit impérativement exister avant qu'on lui assigne un tâches. Ensuite, il faut aussi suivre quelques contraintes : l'≪ID≫ doit être des entier relatif positif, le ≪STATUS≫ doit seulement 1 des 3 valeurs propose (nouveau, en progrès ou terminé).

Noté bien que pour les 3 actions dans le menu qui sont : assignes, modifier, supprimer effectue directement une mise à jour de l'affichage après chaque validation. Donc nous pouvons voir directement le dernier élément ajouter qui est sur la dernière ligne de l'affichage (FIGURE 10).

```
CHOIX:
1
**********************************
MENU ASSIGNATION TACHE
**********************************
SAISIR ID:
5
SAISIR NOM:
Jean
SAISIR DESCRIPTION:
complexite
SAISIR STATUS (nouveau, en progres, termine):
nouveau
SAISIR MEMBRE ID:
1
ID:1 **NOM:tache1 **DESCRIPTION:gla **STATUS:nouveau **MEMBRE ID:1
ID:2 **NOM:tache2 **DESCRIPTION:sma **STATUS:en progres **MEMBRE ID:2
ID:3 **NOM:tache3 **DESCRIPTION:francais **STATUS:termine **MEMBRE ID:2
ID:4 **NOM:tache4 **DESCRIPTION:anglais **STATUS:termine **MEMBRE ID:3
ID:5 **NOM:Jean **DESCRIPTION:complexite **STATUS:nouveau **MEMBRE ID:1
BUILD SUCCESSFUL (total time: 1 minute 22 seconds)
```

FIGURE 10 – Affichage du menu assignation des taches

En ce qui concerne le menu de recherche des tâche ; il existe deux possibilité :

soit par l'≪ID≫ du membre qui permet de lister toutes les tâches effectué par un membre donné (FIGURE 11), soit par le ≪STATUS≫ de chaque tâches. Nous pouvons voir pour le cas de la (FIGURE 11) la liste des tâches effectuer par le membre avec l'identifiant ≪MEMBRE ID≫ = 2.

```
CHOIX:
4
**********************************
MENU RECHERCHE TACHE
**********************************
SAISIR 0 ANNULER ET QUITTER APPLICATION
SAISIR 1 RECHERCHE PAR ID MEMBRE
SAISIR 2 RECHERCHE PAR STATUS
CHOIX:
2
SAISIR STATUS:
termine
ID:3 **NOM:tache3 **DESCRIPTION:francais **ID MEMBRE:2
ID:4 **NOM:tache4 **DESCRIPTION:anglais **ID MEMBRE:3
BUILD SUCCESSFUL (total time: 26 seconds)
```

FIGURE 11 – Affichage du menu recherche des taches par ID du membre

Pour le cas du membre : pas une grande différence la (FIGURE 12) nous montre la présentation du menu principale du membre. Les actions proposés sont identiques (ajouter, modifier, supprimer, recherche et quitter l'application).

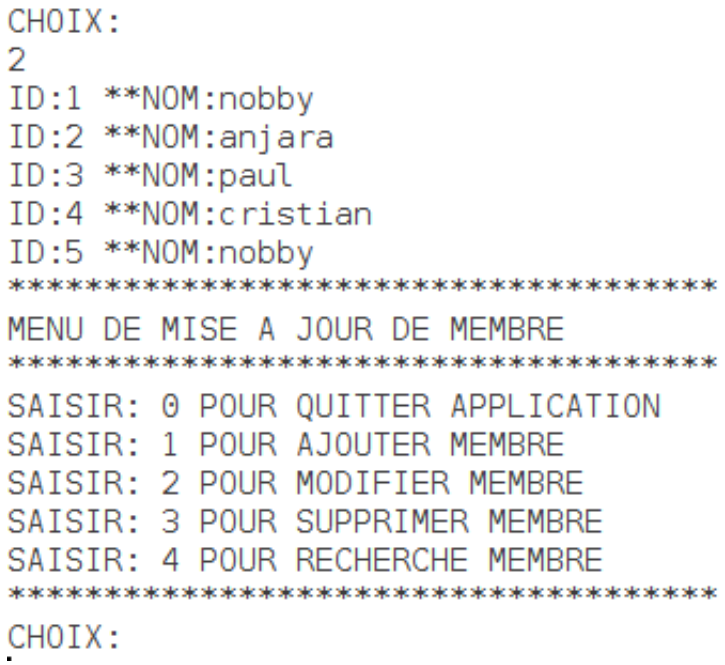

FIGURE 12 – Affichage du menu membre

La (FIGURE 13) présente le menu de modification des membres. Comme nous avons déjà précise ci-dessus, la mise à jour de l'affichage s'effectue directement en bas de la ligne (FIGURE 13).

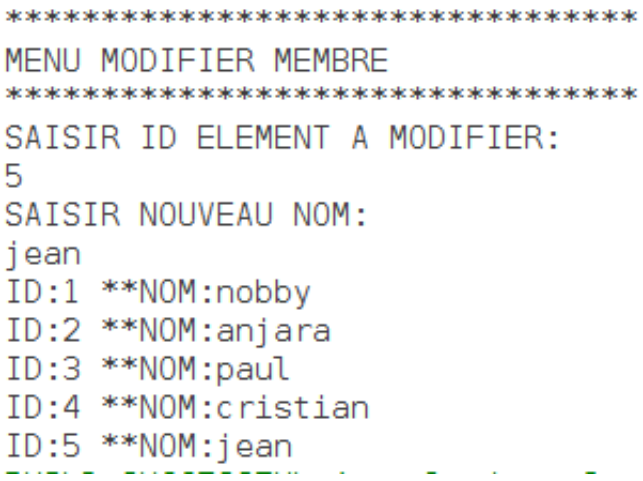

FIGURE 13 – Affichage du menu modification membre

2.5 Test unitaire avec JUNIT

Le test unitaire avec JUNIT permet de vérifier la fonctionnalité des petits portions de code (fonction). Le framework a été écrit par Kent Beck. Depuis

Java 5, plusieurs importantes évolutions ont été apportées à JUnit. [Hunt et Thomas 2003]

Dans le cadre de notre Travaux Pratique (TP), 4 tests ont été effectue : (2 ont été utiliser pour des bonnes valeurs et les 2 autres avec des valeurs erroné pour vérifier la réponse de ses fonctions). La librairie **hamcrest** a été utiliser durant la phase de test. Cette librairie permet accéder facilement à l'intérieur de la liste pour vérifier une valeur.

scénario 1 «créer tâche vrai valeur» : le premier test consiste a créer une tâche, on suite on a vérifier si la valeur qu'on ajouter existe dans la liste des tâche.

scénario 2 «créer tâche faux valeur» : le second test, consiste aussi à créer une tâche. Cependant on a vérifier par une autre valeur si cet élément existe dans la liste des tâche. Ce qui nous donne un erreur au cours du test, ce qui est parfaitement normale étant donne que l'élément n'existe pas dans la liste.

scénario 3 «rechercher tâche vrai valeur» : le troisième test consiste aussi à ajouter aussi un élément dans la base. Ensuite, on effectue une recherche dans la liste. Comme l'élément existe vraiment dans la liste, alors le test à réussi.

scénario 4 «rechercher tâche faux valeur» : le dernier test se base sur le même principe que le test 3. Cependant, on effectue ensuite, un recherche d'un élément qui n'existe pas. Ce qui nous donne un erreur au cours du test, ce qui est parfaitement normale étant donne que l'élément n'existe pas dans la liste.

La (FIGURE 14) nous montre les résultats obtenus au cours du test.

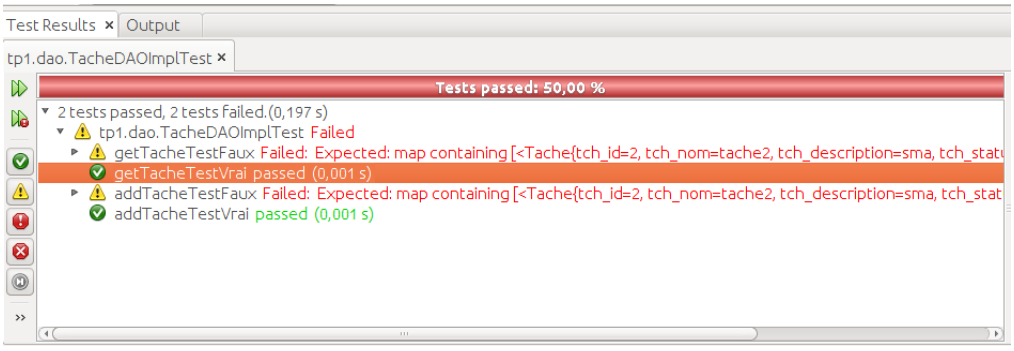

Figure 14 – Résultat du test avec JUNIT

2.6 Test d'acceptation

Le test d'acceptation constitue à une analyse de l'utilisabilité de l'application. En d'autre termes, comment se présente l'application du point de vue de l'utilisateur. Plusieurs tests peuvent être effectués pour valider l'acceptabilité de l'application :

Navigabilité (25%) : Les choix de menu proposés offre une meilleur navigabilité de l'application. De plus, une assistance aide pour le type et la

valeur du l'élément à saisir (FIGURE 15). Donc pour la partie navigabilité, la note attribuée est de 25/25.

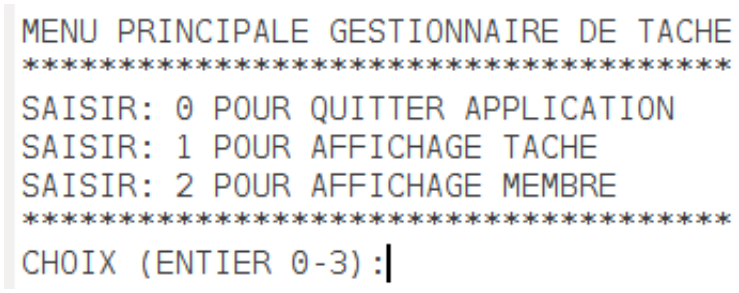

FIGURE 15 – Assistance pour le saisi

Apprentissage (25%) : L'ordonnancement des tâches offre une meilleur apprentissage de l'application. D'abord l'affichage du menu principale, ensuite le menu tâche ou membre. Enfin, chaque action (ajout, modification, suppression ou recherche) peut être effectuer facilement. Cependant, l'interface de présentation console diminue l'apprentissage de l'application. Donc pour la partie apprentissage, la note attribuée est de 20/25.

Visibilité (25%) : L'interface console sans interface graphique diminue la visibilité de l'application. Cependant tous les menu sans très bien ordonné et visible. Donc pour la partie visibilité, la note attribuée est de 18/25.

Assistance (25%) : Dans la partie présentation de l'application et manuel d'utilisation décrit avec détail l'utilisation de l'application 25/25.

Total (100%) : Ce qui nous donne au totale la note de 88%.

Seuil (75%) : Donc on prenant un seuil de 75% qui est largement inférieur à 88% , on peut affirmer que le test d'acception est réussi.

2.7 Validation de l'application

Pour validé notre application, nous avons exécuter plusieurs scénarios et à la fin choisir le scénario optimale.

Scénario 1 : Utilisé une base de données (Mysql), nécessite l'utilisation des librairies, et plus lent en chargement de la base.

Scénario 2 : Utilisé une liste, pas besoin d'utiliser des librairies, moins de temps pour le chargement de l'application. Offre plus de facilité pour le déploiement.

Conclusion : Alors, d'après le ≪scénario 2≫, on remarque une amélioration de la performance. Donc ont peut conclure que notre modèle est validé.

2.8 Déploiement de l'application

L'application a été réaliser avec le moins de contrainte possible. En d'autres termes, pas besoin d'installation d'autres librairies. Donc, pour pouvoir lancer l'application, on importer le projet tous simplement.

Le programme a été déposé sur le plateforme de développement collaboratif et gestionnaire de version github avec l'url : (https ://github.com/anjara4/TP1_GLA).

3 Conclusion

En somme le projet actuel constitue une remise à niveau en Programmation Orienté Objet (POO JAVA). Le travail que nous avons effectué consiste à développer une application de gestionnaire de tâche. En effet, voici une partie de la liste des exigences de l'application : crée une tâche, mise à jour tâche, assigner tâche à un membre et faire une recherche des tâche. Ses exigences ont été satisfaite durant l'étape de conception et de l'implémentation du programme. En outre, une étape de validation et d'acceptation a été effectue à la fin du développement du programme pour vérifier la consistance et validité de notre application.

Les principales avantages de cette application est la facilite de déploiement (sans nécessite d'installer des librairies sauf pour les cas de test unitaire), l'application est multi-plateforme et aussi une grand de facilite d'appréciabilité.

Le travail actuel n'est pas parfait, plusieurs perspectives sont encore à discuté nous pouvons cité quelques une comme la mise en place du 3 couche du modèle MVC. Mais aussi, nous pouvons aussi mettre plusieurs contraintes pendant la saisi des informations par les utilisateurs (regex).

Références

Muller, Pierre-Alain et Nathalie Gaertner (2000). *Modélisation objet avec UML*. T. 514. Eyrolles Paris.

Ledang, Hung (2001). "Des cas d'utilisation à une spécification B". In : *Approches Formelles dans l'Assistance au Développement de Logiciels-AFADl'2001*, 10–p.

Hunt, Andy et Dave Thomas (2003). *Pragmatic unit testing in Java with JUnit*. The Pragmatic Bookshelf.

Marston, Tony (2004). "The model-view-controller (mvc) design pattern for php". In : *Retrieved September* 30, p. 2012.

Martin, Nicolas, Samuel Degrande et Christophe Chaillou (2005). "Une utilisation du modèle mvc pour une plate-forme de travail virtuel". In : *Proceedings of the 17th Conference on l'Interaction Homme-Machine*. ACM, p. 131–138.

Charroux, Benoît (2008). "UML 2.0". In :

Audibert, Laurent (2009). *UML 2 : de l'apprentissage à la pratique : présentation des diagrammes UML (cas d'utilisation, classes, objets, états-transitions, activités, structures composites, communication, séquence, composants, déploiement), langage de contraintes OCL, introduction aux patrons de conception (design patterns), mise en oeuvre d'UML*. Ellipses.

Articles only

Marston, Tony (2004). "The model-view-controller (mvc) design pattern for php". In : *Retrieved September* 30, p. 2012.

Charroux, Benoît (2008). "UML 2.0". In :

Books only

Muller, Pierre-Alain et Nathalie Gaertner (2000). *Modélisation objet avec UML*. T. 514. Eyrolles Paris.

Hunt, Andy et Dave Thomas (2003). *Pragmatic unit testing in Java with JUnit*. The Pragmatic Bookshelf.

Audibert, Laurent (2009). *UML 2 : de l'apprentissage à la pratique : présentation des diagrammes UML (cas d'utilisation, classes, objets, états-transitions, activités, structures composites, communication, séquence, composants, déploiement), langage de contraintes OCL, introduction aux patrons de conception (design patterns), mise en oeuvre d'UML*. Ellipses.