Université du Québec
École de technologie supérieure

Département de génie logiciel et des T.I.

Rapport de Laboratoire

Numéro du laboratoire	1
Étudiant(s)	Gislain Armand Maxime Bédard Xavier Drdak
Code(s) permanent(s)	ARMG14059104 BEDM19129002 DRDX16039107
Numéro d'équipe	03
Cours	LOG430
Session	Automne 2014
Groupe	01
Chargé(e) de laboratoire	Samir Djeffal
Date	1 Octobre 2014

Table de matières

Introduction

Lors de la conception d'un logiciel, tout comme la réalisation de tout projet en ingénierie, il est primordial d'avoir une structure bien établie. Dans le domaine du génie logiciel, il existe une multitude de styles architecturaux afin de spécifier comment doit être conçu un logiciel. Dans le cadre de ce laboratoire, le style d'architecture en couche est étudié au travers de l'analyse d'une application existante. L'application à l'étude permet d'assigner des ressources à des projets de développement logiciel.

Dans un premier temps, l'analyse de l'architecture existante est effectuée. À partir des différents outils de travaux disponibles, il est possible d'observer les différentes relations entre les composantes du logiciel à notre disposition.

Dans un deuxième temps, des fonctionnalités et des modifications sont apportées à l'application existante afin de répondre aux nouveaux besoins.

Dans un troisième temps, un plan de test est rédigé afin de pouvoir valider le bon fonctionnement des diverses fonctionnalités ajoutées par rapport aux nouvelles exigences. Par la suite, une nouvelle analyse est effectuée afin de pouvoir observer les répercussions des différentes modifications sur l'architecture existante.

Enfin, une discussion est portée sur les améliorations qui peuvent être apportées et permet de faire un retour sur les différents éléments observés tout au long de ce laboratoire.

Sommaire de l’implémentation

Modification 1

La première modification doit implémenter une option qui permet d’afficher les ressources qui étaient déjà assignées à un projet avant l’exécution du système. Pour se faire, il est important d’observer les fonctionnalités déjà offertes par le système. Tout d’abord, la liste des projets et la liste des ressources des sessions antérieures sont chargées en mémoire au lancement de l’application. Ces listes proviennent des fichiers projects_test.txt et resources_test.txt. De plus, les anciens projets sont assignés aux différentes ressources, mais uniquement par leur numéro de projet.

Ainsi, la modification à effectuer est d’établir la relation entre les numéros de projets assignés à la ressource au projet lui-même. De plus, il est nécessaire d’ajouter les diverses fonctions pour permettre l’affichage détaillé des projets pour les sessions précédentes.

Afin d’établir la relation entre les projets et les ressources, il a été d’une simplicité alarmante d’ajouter uniquement une dépendance à la liste des projets lors de la création d’une ressource. En effet, lors de la lecture du fichier pour récupérer les ressources des sessions antérieures, la liste des projets est directement transférée en référence et le lien est établi lors de la lecture du numéro de projet. Cette option a été retenue particulièrement due à sa simplicité et a la rapidité de mise en œuvre.

Par la suite, comme les modifications 2 et 3, une fonction d’affichage a été ajoutée afin de pouvoir récupérer les différents projets antérieurs et de les afficher à l’écran. Cette fonctionnalité nécessite l’utilisateur pour lequel les projets antérieurs doivent être affichés et les affiches à l’écran exactement comme les autres fonctionnalités existantes qui affichent la liste des projets.

Modification 2

La deuxième modification de l'application consiste à l’ajout d’une option permettant d'afficher tous les rôles ayant été assignés à un projet spécifique. L'affichage des rôles est effectué en deux parties: les rôles ayant été assignés avant l'exécution courante et les rôles assignés durant l'exécution courante.

Afin d’implémenter cette modification, l’utilisateur doit préalablement choisir un projet. Une fois le projet est sélectionné, ce dernier est envoyé à la classe *Display,* où la logique est implémentée. En plus de recevoir le projet, la méthode a besoin de la liste des ressources pour être en mesure de distinguer entre une ressource ayant déjà été assignée et une ressource ayant été assignée durant l’exécution de l’application.

Or, pour chaque ressource contenue dans la liste, nous regardons si elle contient le projet dans sa liste des projets assignés précédemment. Si cette condition est vraie, nous regardons si le rôle de la ressource a déjà été affiché, sinon nous l’affichons.

Pour les rôles assignés à un projet durant l’exécution, c’est sensiblement la même logique. Par contre, au lieu de regarder dans la liste des projets assignés précédemment, nous regardons dans la liste des projets assignés durant l’exécution.

Dans les deux cas, s’il n’y a pas de rôles, un message s’affiche dans le but d’indiquer à l’utilisateur qu'il n'y a aucun rôle à afficher

Modification 3

La troisième modification consiste à implémenter un système pour attribuer des projets à une ressource. Chaque projet possède une priorité de travail et occupe une ressource à un certain taux. Il doit être impossible pour l'utilisateur de surcharger une ressource. De plus, il faut prendre en considération la durée du projet. En autre mot, un projet qui est terminé ne doit plus consommer une ressource. Afin d'implémenter cette fonctionnalité, il est nécessaire de modifier les classes *Project* et *Ressource*.

La classe *Project* doit être en mesure de connaitre sa priorité, et déterminer si ce dernier est complété. En effet, un *Project* conserve initialement sa priorité et la date de début et fin du projet. En utilisant ce paramètre de priorité, la date présente, ainsi que la date de début et fin du projet, il est possible de calculer le taux d'occupation, soit la charge totale d'un projet.

La Classe *Ressource* contient une liste de projets. Avec les modifications apportées à la classe précédente, il est possible pour une ressource de calculer son taux d'occupation. Lorsqu'on désire ajouter un nouveau projet à une ressource, l'application parcourt la liste des projets récemment assignés, ainsi que ceux assignés dans le passé, puis additionne la charge de travail total.

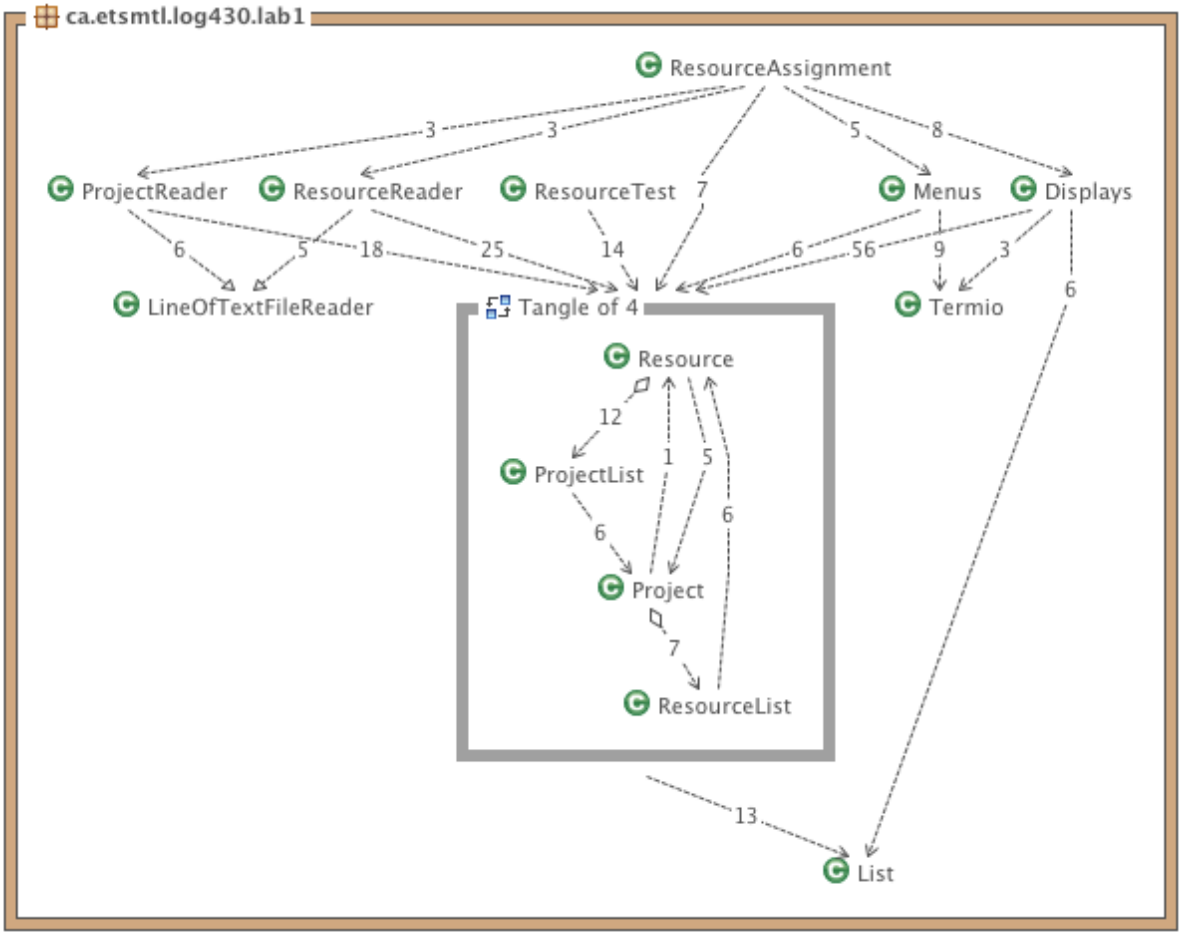
Avec ces modifications apportées, il est maintenant possible d'attribuer un projet à une ressource. Lorsque l'utilisateur désire effectuer cette opération, l'application vérifie si la charge de travail de la ressource demandée. Si cette charge de travail, additionné avec le nouveau projet désiré, dépasse 100 %, l'application envoie un signal à la classe *ResourceAssignment* pour l'aviser d'abandonner l'opération d'affectation de projet vers une ressource et d'afficher un message d'erreur. Cependant, si la charge de travail totale est inférieure ou égale à 100 %, l'opération d'affectation de projet est exécutée et un message de succès est affiché.

Analyse architecturale

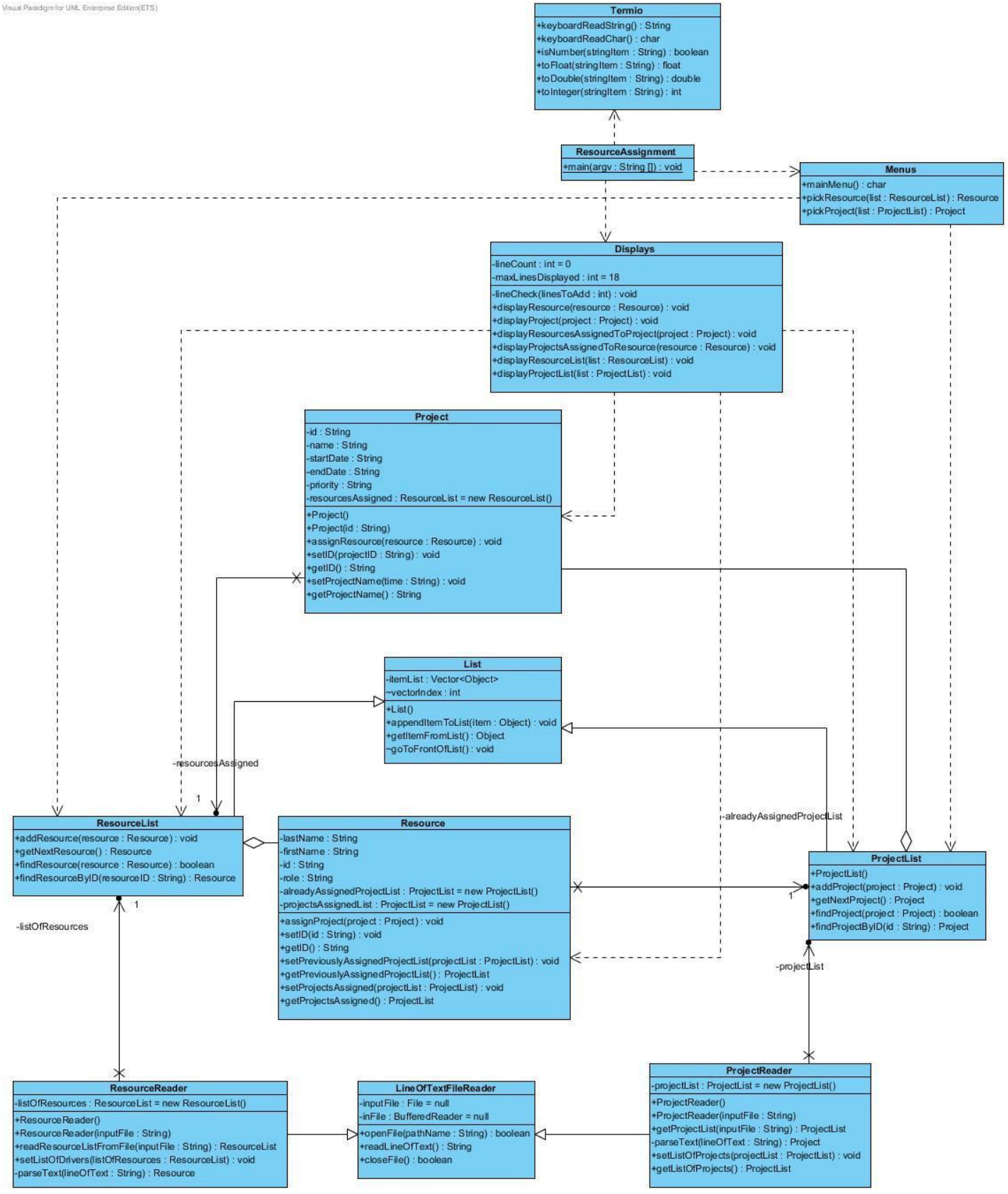
Vue Architecturale

Système initial - Vue de composition

Étant donné qu'il n'y a aucun regroupement des classes dans les paquets, tout est désorganisé. En effet, il est très difficile de voir l'architecture en couche de l'application.

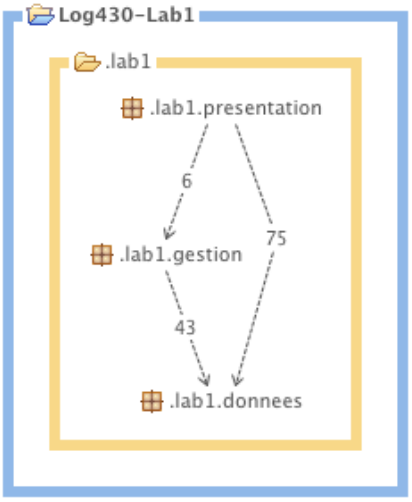


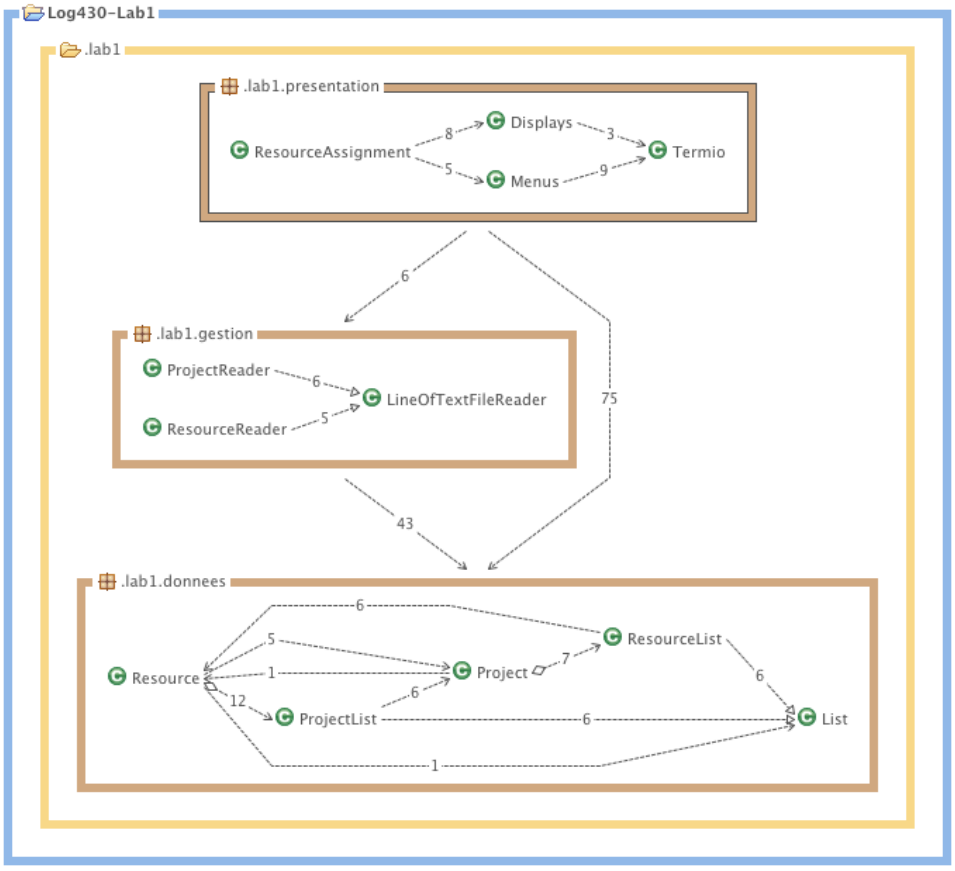
Système initial - Diagramme de classe



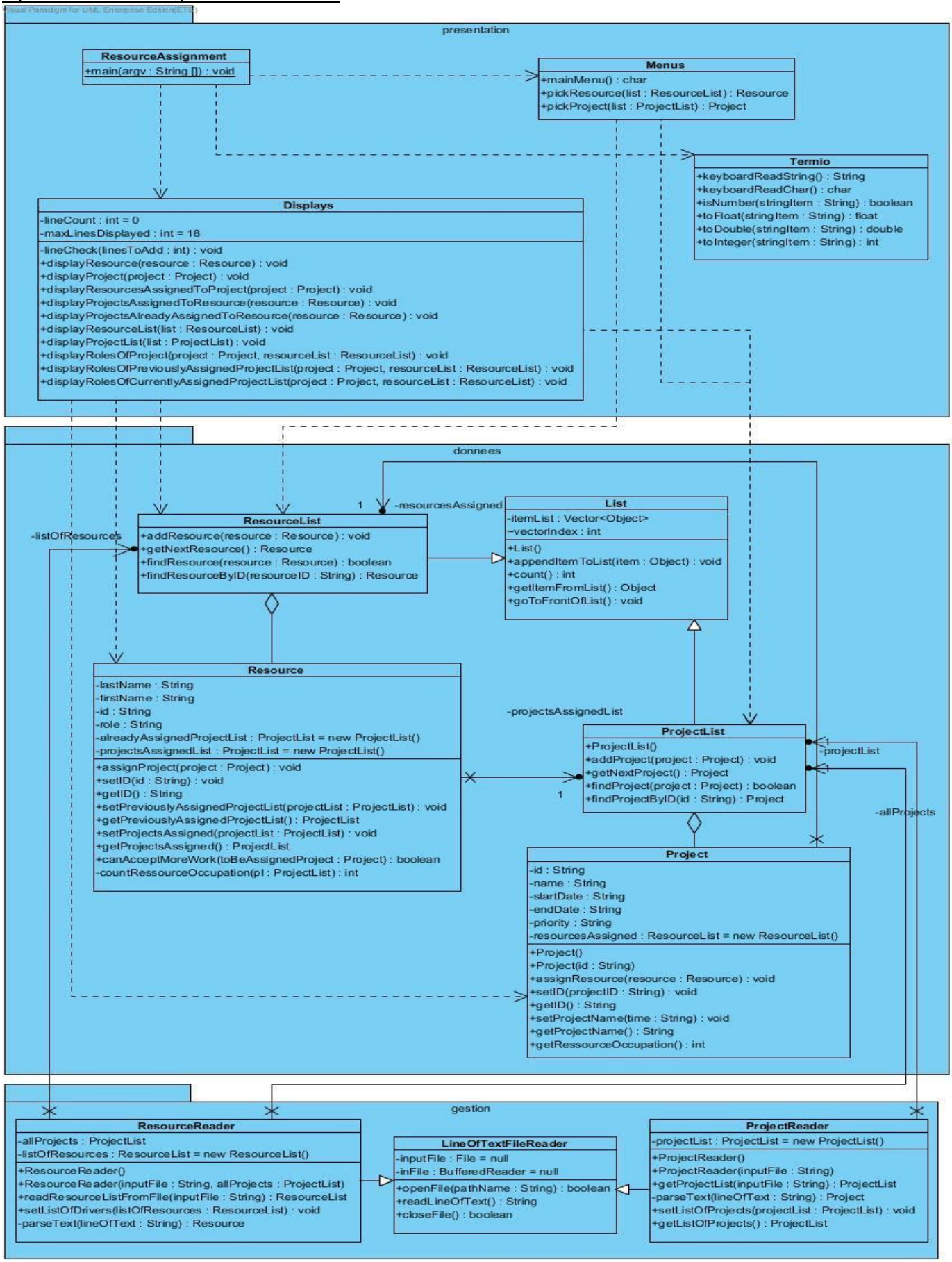
Système final - Vue de composition

Voici la vue de composition générée par Stan4j. Il est possible de remarquer que l'application respecte entièrement l'architecture en couche.





Système final - Diagramme de classe



Mappage des classes

Avec l'utilitaire d'analyse de code STAN4j, il est possible de visualiser l'architecture du code source. Notre application, suivant les principes de l'architecture en couche, possède trois couches différentes : présentation, gestion et données.

Il va de soi que les classes qui gèrent l'affichage de l'information sont contenues dans la couche de présentation, les classes ayant la logique de l'entreprise se situent dans la couche gestion et les conteneurs d'informations sont situés dans la couche des données.
L'architecture en couche permet aux paquets situant dans le haut de la hiérarchie de communiquer avec ceux qui sont dans le bas. Cependant, les paquets qui sont dans le bas de la hiérarchie cette architecture ne peuvent pas utiliser avec les ressources des niveaux supérieurs. Sachant cela, il est possible d'effectuer un mappage des classes à l'architecture présent.

Les classes ResourceAssignement, Displays, Menus et Termio sont situés dans la couche de présentation. Ces derniers sont responsables d'afficher les projets et les ressources disponibles à cette application, ainsi qu'offrir de l'interactivité auprès des utilisateurs. Or, ces classes doivent avoir la possibilité d'accéder à toutes les données de l'application. Par conséquent, ils doivent aussi avoir accès à la couche de gestion pour lire les fichiers contenant les projets et ressources pour être en mesure d'afficher les informations de texte brut.

Les classes ProjectReader, RessourceReader et LineOfTextReader sont des membres de la couche de gestion. En effet, ces classes sont utilisées pour récupérer et traiter les informations des fichiers contenant les projets et ressources. Ainsi, ils doivent avoir accès à la couche des données pour être en mesure de bâtir des listes et des objets correspondants. Cependant, la couche gestion n'a pas d'accès à la couche supérieure, la présentation des données.

Les classes Ressource, ProjectList, Project, RessourceList et List sont contenues dans la couche donnée. Leur seule responsabilité est de contenir l'information. Ils ne peuvent pas afficher leur information ou se copuler avec des données pertinentes. Ces tâches sont déléguées à la couche de présentation et gestion respectivement. Ainsi, à cause de la nature de la structure de ce projet, les diverses classes qui sont contenues dans la couche de données peuvent se communiquer entre eux. Or, ces derniers ont la possibilité d'effectuer des traitements d'information (par exemple, comptabiliser la charge de travail) sans déléguer les tâches à la couche de gestion. Malgré ce, les classes appartenant dans la couche gestion sont entièrement indépendantes des autres couches.

Interprétation des vues de compositions et améliorations

La vue de composition permet d'identifier rapidement la dépendance entre les classes et les paquets. En effet, tous les liens entre les divers modules sont affichés, incluant la direction des dépendances.

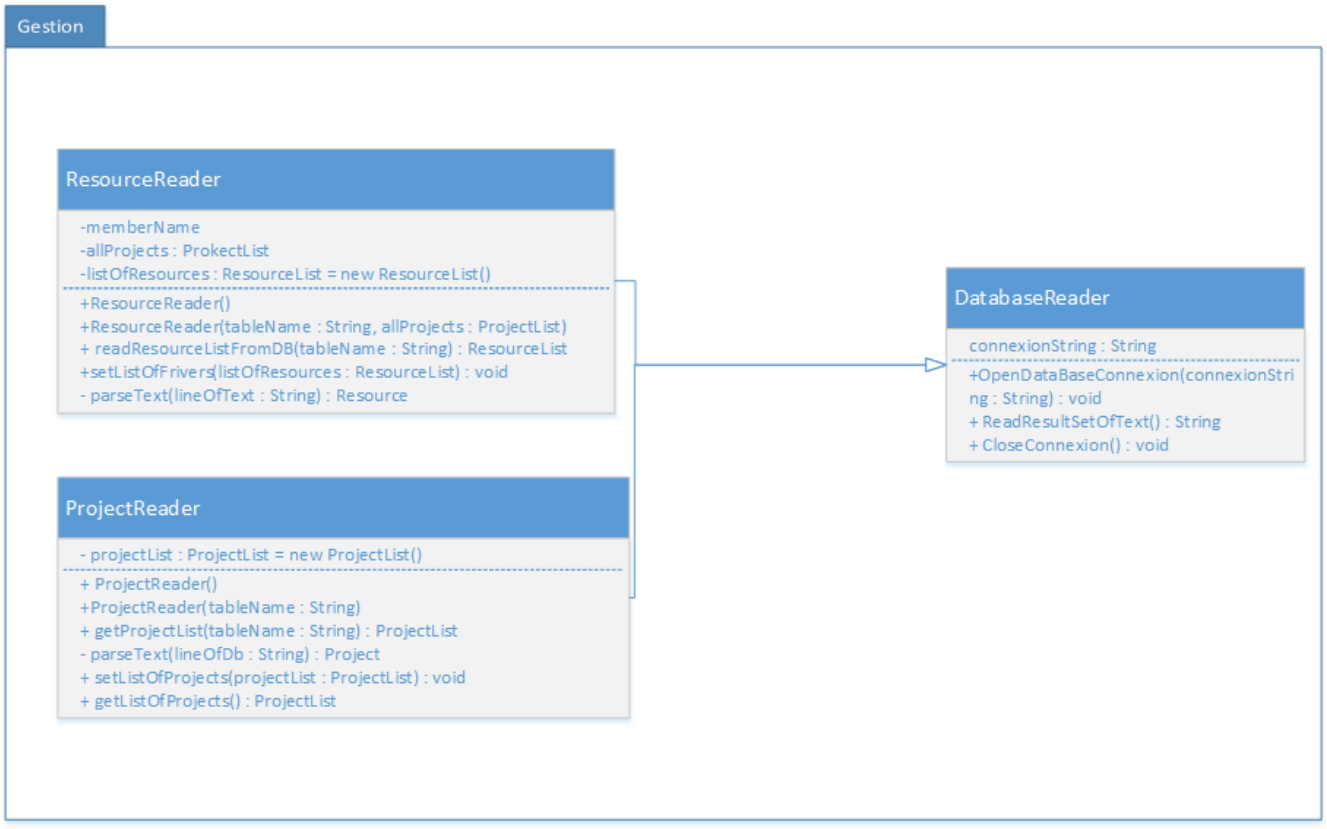
Dans l'utilitaire d'analyse de code stan4j, lorsqu'une vue de composition est générée, les classes sont affichées avec un petit symbole « C » suivi par leur nom. Ces classes sont habituellement accompagnées d'une flèche qui lie avec une ou plusieurs classes. Parallèlement, les paquets encadrent ces classes et possèdent aussi des flèches qui lient des paquets entre eux. Ces flèches, que ce soit des classes ou de paquets, possèdent une direction. Cette direction correspond au sens de la dépendance.

Par exemple, dans la vue de composition du système finale, le paquet lab1.gestion possède deux flèches. Une provenant de lab1.presentation, tandis que l'autre flèche pointe vers lab1.donnes. Ceci indique que lab1.presentation est utilisé par des classes du paquet de lab1.presentation et qu'il utilise les classes provenant du paquet lab1.donnes.

En connaissant les liaisons et les dépendances inter-paquets et inter-classes, il est possible de rapidement identifier les problèmes architecturaux du système. En utilisant les liaisons des dépendances comme point de référence, il est possible de modifier le code source pour s'assurer d'une cohérence architecturale. Soit en déplaçant les classes dans un paquet ou en modifiant les classes elles-mêmes pour changer leur composition.

Modification du système pour lire une base de donné

Afin d'être en mesure de récupérer les informations contenues dans une base de données, un réusinage de code est requis. Cette action aura peu de répercussions sur architecture actuelle. Dans la couche de gestion, il est nécessaire de créer une nouvelle classe, soit «*DatabaseReader*». Ce dernier aura la responsabilité de récupérer les informations nécessaires dans la base de données, et non dans un fichier texte. Évidemment, les données devront être dans le même format que celles dans le fichier texte pour simplifier le réusinage de code. Les deux classes qui s'occupent de charger les objets, «*ResourceReader*» et «*ProjectReader*», vont étendre cette nouvelle classe afin d'avoir accès aux méthodes de gestion de la connexion à la base de données. Dans les deux cas, il doit y avoir une ouverture de connexion, une lecture des informations, puis une fermeture de la connexion.



Pour créer une implémentation plus robuste, il faut effectuer des modifications majeures au niveau de la couche gestion. Tout d'abord, la classe «*DatabaseReader* »doit être remplacé la classe «*LineOfTextFileReader* », étant donné que l'information ne provient plus d'un fichier texte. Cependant, l'application sera en mesure de gérer les bases de données relationnelles, ainsi qu'importer toutes les ressources et les projets avec tous les liens faits vers les divers objets Java.

Essentiellement, il faut créer un «*mapping* » entre les informations provenant de la base de données et les objets de notre application. Les classes «*ResourceReader* » et «*ProjectReader* » nécessite d'une modification importante du code source, dans le but d'accommoder des informations provenant d'une base de données. Pour ce faire, il serait optimal que ces derniers n'étendent pas la classe «*DatabaseReader*». Plutôt, elles devraient demander à «*DatabaseReader*» de leur retourner un «*ResultSet*» du contenu de la table. «*ResourceReader*» et «*ProjectReader*» auront la responsabilité de créer leurs objets respective en exécutant différentes requêtes à la base de données en passant les requêtes par le «*DatabaseReader*».

Discussion

Tout au long de ce laboratoire, plusieurs difficultés ont été rencontrées. Nous avons eu beaucoup de difficulté à établir la relation entre les projets et les ressources des sessions antérieurs. En effet, nous avons décidé d'établir la relation directement lors de la lecture du fichier de ressource, cependant il a été possible d'observer une augmentation directe du couplage entre les deux classes. Nous avons opté particulièrement pour cette solution, car celle-ci permet d'ajouter rapidement la fonctionnalité désirée tout en conservant l'architecture en couche initiale proposée. Cette manipulation a augmenté le couplage entre les deux classes, car une dépendance entre la classe « *ProjectList* » est ajoutée à la classe « *ResourceReader* ».

Une autre difficulté notable que nous avons rencontrée lors de la conception de ce laboratoire a été la rédaction d'un plan de test. Il a été difficile de rédiger un plan de test efficace qui permet de répondre au plus grand nombre de cas de tests possible. De plus, la solution la plus évidente pour nous était de concevoir des tests unitaires qui permettent de valider les différentes modifications. Afin de répondre aux exigences de l'énoncé, nous avons décidé de conserver les différents tests unitaires qui ont été rédigés, mais nous avons aussi rédigé, au format texte, un plan de test concis, ainsi que des fichiers de tests permettant de répondre aux différents cas de tests qui ont été formulés.

Sous un tout autre ordre d'idée, une multitude d'améliorations peuvent être effectuées afin de mieux répondre aux différents attributs de qualités du logiciel. Tel que mentionné dans la section précédente, l'ajout d'une base de données pour gérer les données est une modification qui apparait évidente.

Une autre modification d'une semblance évidente est le retrait ou l'ajout de fonctionnalités à la classe List. En effet, cette classe encapsule un vecteur de capacité fixe et masque une multitude de fonctionnalités. L'utilisation de cette liste rend difficile le parcours des différents éléments et a un impact direct sur la clarté et la maintenabilité du code source. Un exemple de modification serait tout d'abord d'implanter l'interface « *Iterable* » ou d'étendre la classe « List<E> ». Ces modifications permettent d'implémenter différentes fonctions qui facilitent le parcours de la liste et qui permettent d'éviter de répéter le même genre d'opération à plusieurs reprises.

Conclusion

En somme, ce laboratoire a permis de visualiser concrètement les avantages et les désavantages que peut apporter l'architecture en couche. Cette séparation distincte permet de facilement repérer les différentes composantes du système et de mieux comprendre l'interaction entre ces différentes composantes. De plus, grâce à l'utilitaire stan4j, il est extrêmement facile d'observer le couplage entre les classes, les différentes relations, la complexité cyclomatique et bien plus. Ces différentes mesures permettent de simplifier la conception de l'application de manière substantielle, ce qui permet de mieux répondre aux différents attributs de qualité du système. Enfin, il apparaît évident que plusieurs modifications peuvent être apportées afin d'améliorer le système existant, mais existe-t-il un modèle architectural ou une combinaison de modèles plus appropriés que le modèle par couche pour ce type de système?

Annexe

Jeu de Test

Préface au tests

Étant donné qu'il y a plusieurs cas possibles, il est nécessaire d'effectuer de multiples tests. Les fichiers de tests utilisés sont :

- «projects_test.txt», le fichier des ressources
- «resources_test.txt», le fichier des projets.

Afin d'être en mesure d'effectuer les tests indiqués dans ce document, il est important de lancer l'application avec les fichiers fournis.

```
javac RessourceAssignement projects_test.txt resources_test.txt
```

Attention : Toutes les entrées sont sensibles à la case!

Modification 1

La première modification de l'application est d'ajouter l'association entre les projets des sessions antécédentes. Les tests suivants permettent de valider à partir de l'affichage que les relations aient bel et bien été établies.

Premier test :

Une ressource qui n'est pas associée à un projet dans la session antérieure n'affiche aucun projet.

Opérations à effectuer :

1. Sélectionner l'option: 6)
2. Entrer la ressource suivante: R1

L'application devrait afficher la liste des projets pour la ressource R1. Cependant, cette liste devrait être vide, car aucun projet n'est assigné dans le fichier resources.txt.

Deuxième test :

Une ressource est associée à un seul projet dans la session antérieure, le système affiche uniquement ce projet.

Opérations à effectuer :

1. Sélectionner l'option: 6)
2. Entrer la ressource suivante : R002

L'application devrait afficher le projet P002 assigné précédemment à la ressource R002. Les informations complètes pour ce projet devraient être les suivantes : P002 Projet2 2013-01-05 2013-07-22 M.

Troisième test :

Une ressource est associée à plusieurs projets dans la session antérieure et les affiches tous.

Opérations à effectuer:

1. Sélectionner l'option: 6)
2. Entrer la ressource suivante : R001

L'application devrait afficher le projet P001 et P003 assigné à la ressource R001. Les informations complètes pour les projets sont les suivantes : P001 Projet1 2012-12-20 2013-05-18 H, P003 Projet3 2013-10-01 2014-03-01 L.

Modification 2

La deuxième modification de l'application est d'ajouter une option qui permet d'afficher tous les rôles ayant été assignés à un projet spécifique. L'affichage des rôles se fait en deux parties; les rôles ayant été assignés avant l'exécution courante et les rôles assignés durant l'exécution courante.

Premier test :

Un projet qui ne contient pas de ressources ne devrait pas afficher de rôles.
Dans le contexte suivant, le projet T2P2 ne contient aucune ressource. Alors il est normal qu'il n'y ait aucun rôle à afficher.

Opérations à effectuer :

1. Sélectionnez l'option : 7)
2. Une liste de projets s'afficher et il est important d'aller jusqu'en bas de la liste en appuyant sur "Enter".
3. Entrez le projet suivant: T2P2

L'application devrait mentionner qu'il n'y a aucun rôle à afficher pour les rôles ayant été assignés avant l'exécution courante et les rôles assignés durant l'exécution courante.

Deuxième test :

Un projet qui ne contient aucun rôle dans le passé et qui s'en fait assigner devrait afficher la liste des rôles assignés durant l'exécution courante.
Dans le contexte suivant, le projet T2P2 ne contient aucune ressource. Nous allons donc lui en assigner deux et de s'assurer qu'il va afficher les deux rôles dans la section des rôles assignés durant l'exécution courante.

Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante: R001
3. Une liste de projets s'afficher et il est important d'aller jusqu'en bas de la liste en appuyant sur "Enter".
4. Entrez le projet suivant: T2P2
5. Répétez les étapes 1 à 4, en conservant le projet T2P2, mais avec la ressource R1.
6. Sélectionnez l’option : 7)
7. Une liste de projets s'afficher et il est important d'aller jusqu'en bas de la liste en appuyant sur "Enter".
8. Entrez le projet suivant: T2P2

L'application devrait mentionner qu'il n'y a aucun rôle à afficher pour les rôles ayant été assignés avant l'exécution courante, mais il devrait y avoir deux rôles assignés durant l'exécution courante, soit ANA et TST.

Troisième test :

Un projet qui ne contient aucun rôle dans le passé et qui s'en fait assigner devrait afficher la liste des rôles assignés durant l'exécution courante.
Dans le contexte suivant, le projet T2P2 ne contient aucune ressource. Nous allons donc lui en assigner deux qui contiennent le même rôle et s'assurer qu'il va afficher qu'un rôle dans la section des rôles assignés durant l'exécution courante.

Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante: R1
3. Une liste de projets s'afficher et il est important d'aller jusqu'en bas de la liste en appuyant sur "Enter".
4. Entrez le projet suivant: T2P2
5. Répétez les étapes 1 à 4, en conservant le projet T2P2, mais avec la ressource R2.
6. Sélectionnez l'option : 7)
7. Une liste de projets s'afficher et il est important d'aller jusqu'en bas de la liste en appuyant sur "Enter".
8. Entrez le projet suivant: T2P2

L'application devrait mentionner qu'il n'y a aucun rôle à afficher pour les rôles ayant été assignés avant l'exécution courante, mais il devrait y avoir un rôle assigné durant l'exécution courante, soit TST.

Quatrième test :

Un projet qui contient des ressources d'assignées avant l'exécution courante devrait les afficher dans la section appropriée.
Dans le contexte suivant, le projet P004 contient deux ressources qui ont chacun un rôle différent.

Opérations à effectuer :

1. Sélectionnez l'option : 7)
2. Une liste de projets s'afficher et il est important d'aller jusqu'en bas de la liste en appuyant sur "Enter".

3. Entrer le projet suivant: P004 L'application devrait mentionner qu'il n'y a aucun rôle à afficher pour les rôles assignés durant l'exécution courante, mais deux rôles dans la section des rôles assignés avant l'exécution courante, soit PRG et TST.

Modification 3

La troisième modification de l'application doit permettre à l'usager d'attribuer un projet à une ressource. Chaque projet a un niveau de charge et il y a trois types de priorités pour un projet.

- 'L' : une charge de travail léger (25% des capacités d'une ressource).
- 'M' : une charge de travail moyenne de (50% des capacités d'une ressource).
- 'H' : une charge de travail élevé de (100% des capacités d'une ressource).

Un usager ne peut avoir de nouveau projet si la totale de la charge de travail présent dépasse 100%.

Premier test :

Une ressource (R1) n'ayant aucun projet d'attribué tente d'accepter 5 nouveaux projets (T1P1, T1P2, T1P3, T1P4, T1P5). Tous ces projets ont une priorité 'L'. Ainsi, l'application devrait bloquer le cinquième ajout, car la charge de travail dépasse 100%.

En effet, un projet ayant une priorité 'L' occupe une ressource à 25 % de capacité. Or, dans le contexte suivant, la ressource ne peut seulement accepter 4 nouveaux projets possédant une priorité 'L', pour une charge de travail total de 100%. Tout autre projet devrait être impossible à attribuer à une ressource.

Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante R1
3. Entrer le projet suivant: T1P1
4. Répéter les étapes 1 à 3, en conservant la ressource (R1), mais avec le projet T1P2.
5. Répéter les étapes 1 à 3, en conservant la ressource (R1), mais avec le projet T1P3.
6. Répéter les étapes 1 à 3, en conservant la ressource (R1), mais avec le projet T1P4.
7. Répéter les étapes 1 à 3, en conservant la ressource (R1), mais avec le projet T1P5.

L'application devrait afficher une erreur de ressource surchargée à la fin de l'opération 7.

Deuxième test :

Une ressource (R2) n'ayant aucun projet d'attribué tente d'accepter 3 nouveaux projets (T2P1, T2P2, T2P3). Ces projets ont tous une priorité 'M' de 50%. L'application devrait bloquer l'ajout du troisième projet.

Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante R2
3. Entrer le projet suivant: T2P1
4. Répéter les étapes 1 à 3, en conservant la ressource (R2), mais avec le projet T1P2.
5. Répéter les étapes 1 à 3, en conservant la ressource (R2), mais avec le projet T1P3.

L'application devrait afficher une erreur de ressource surchargée à la fin de l'opération 5.

Troisième test :

Une ressource (R3) n'ayant aucun projet d'attribué tente d'accepter 2 nouveaux projets (T3P1, T3P2). Ces projets ont une priorité 'H'. L'application devrait bloquer l'ajout du deuxième projet, car la charge de travail dépasse 100%.
Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante R3
3. Entrer le projet suivant: T3P1
4. Répéter les étapes 1 à 3, en conservant la ressource (R3), mais avec le projet T3P2.

L'application devrait afficher une erreur de ressource surchargée à la fin de l'opération 4.

Quatrième test :

Une ressource (R4), ayant déjà un projet (T4P1) ayant une priorité ‘L’ tente d’ajouter 2 projets (T4P2, T4P3) ayant une priorité ‘M’. L’application doit bloquer l’ajout du deuxième projet, car la charge de travail dépasse 100%

Opérations à effectuer :

1. Sélectionner l’option : 5)
2. Entrer la ressource suivante R4
3. Entrer le projet suivant: T4P2
4. Répéter les étapes 1 à 3, en conservant la ressource (R4), mais avec le projet T4P3.

L’application devrait afficher une erreur de ressource surchargée à la fin de l’opération 4.

Cinquième test :

Une ressource (R5) a déjà un projet (T5PE) ayant une charge ‘H’. Cependant, ce projet est expiré. L’utilisateur souhaite d’ajouter 1 nouveau projet (T5P1) ayant une priorité H. L’application devrait permettre cette opération, car le projet T5PE est expiré et la ressource (R5) devrait être libérée de cette ancienne charge de travail.

Opérations à effectuer :

1. Sélectionner l’option : 5)
2. Entrer la ressource suivante R5.
3. Entrer le projet suivant: T5P1.

Il ne devrait avoir aucun message d’erreur à la fin de l’opération 3.

Sixième test :

Une ressource (R6) n'ayant aucun projet d'attribué tente d'accepter 3 nouveaux projets (T6P1, T6P2, T6P3). Ces projets ont une respectivement une priorité 'L', 'M' et 'H' L'application devrait bloquer l'ajout du troisième projet.

Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante: 6.
3. Entrer le projet suivant : T6P1
4. Répéter les étapes 1 à 3, en conservant la ressource (R6), mais avec le projet T6P2
5. Répéter les étapes 1 à 3, en conservant la ressource (R6), mais avec le projet T6P3

L'application devrait afficher une erreur de ressource surchargée à la fin de l'opération 5.

Septième test :

Une ressource (R7), ayant déjà un projet (T7P1) ayant une priorité 'M' tente d'ajouter 2 projets (T7P2, T7P3) ayant une priorité 'M'. L'application doit bloquer l'ajout du deuxième projet, car la charge de travail dépasse 100%

Opérations à effectuer :

1. Sélectionner l'option : 5)
2. Entrer la ressource suivante R7
3. Entrer le projet suivant: T7P2
4. Répéter les étapes 1 à 3, en conservant la ressource (R7), mais avec le projet T7P3.

L'application devrait afficher une erreur de ressource surchargée à la fin de l'opération 4.

Huitième test :

Une ressource (R8) a déjà deux projets (T8PE, T8P1). Le projet T8PE ayant une priorité ‘M’ est expiré. Le projet T8P1 est encore actif et possède une priorité ‘L’. L’utilisateur souhaite d’ajouter un nouveau projet (T8P2) ayant une priorité ’M’. L’application devrait permettre cette opération, car le projet T5PE est expiré et la ressource (R8) devrait être libérée de cette ancienne charge de travail, d’autant plus que la charge de travail de dépasse pas 100%.

Opérations à effectuer :

1. Sélectionner l’option : 5)
2. Entrer la ressource suivante : R8.
3. Entrer le projet suivant: T8P2.

Il ne devrait avoir aucun message d’erreur à la fin de l’opération 3. En effet, la charge totale des deux projets (T8P1, T8P2) est égale à 75%.