Cahier des charges : gestion de projets agiles

Programmation d'Algorithmes Distribués (PAD)

Benoit Bernardin
Rachid Bouherrou
Bruno Juillard
Kaba Kabinè
Sid-ahmed Anouar Lekhal

2012 / 2013

Sommaire

Table des illustrations

Nous débuterons par présenter le contexte du projet : son environnement et comment nous sommes arrivés à ce sujet. Nous présenterons ensuite, le cahier des charges.

PRÉSENTATION DU PROJET

A propos de la gestion de projet agile

Un projet est la réalisation d'un ensemble d'activités ou de fonctionnalités, devant répondre à un besoin spécifique, dans un délai fixé et avec une enveloppe budgétaire allouée. La gestion de projet est l'action de piloter une telle réalisation en la découpant par tâche, en l'organisant, en la planifiant et en mobilisant des ressources identifiées (humaines, matérielles, financières…). Il existe aujourd'hui de multiples façons de conduire un projet mais on distingue notamment les méthodes traditionnelles et les méthodes agiles.

Les méthodes agiles, bien que pouvant être appliquées à divers projets, sont aujourd'hui très répandues en Informatique et notamment dans le développement logiciel. Elles se veulent beaucoup plus pragmatiques que les méthodes traditionnelles en impliquant au maximum le demandeur et en permettant une grande réactivité et une forte adaptation à ses demandes. Ces méthodes reposent sur une structure (cycle de développement) commune (itérative, incrémentale et adaptative) et quatre valeurs communes déclinées en douze principes fondamentaux.

Les quatre valeurs communes :

- L'équipe « *Les individus et leurs interactions, plus que les processus et les outils* »
- L'application « *Des logiciels opérationnels, plus qu'une documentation exhaustive* »
- La collaboration « *La collaboration avec les clients, plus que la négociation contractuelle* »
- L'acceptation du changement « *L'adaptation au changement, plus que le suivi d'un plan* »

Les douze principes fondamentaux :

- La plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à forte valeur ajoutée.
- Le changement est accepté, même tardivement dans le développement, car les processus agiles exploitent le changement comme avantage compétitif pour le client.
- La livraison s'applique à une application fonctionnelle, toutes les deux semaines à deux mois, avec une préférence pour la période la plus courte.
- Le métier et les développeurs doivent collaborer régulièrement et de préférence quotidiennement au projet.
- Le projet doit impliquer des personnes motivées. Donnez-leur l'environnement et le soutien dont elles ont besoin et faites leur confiance quant au respect des objectifs.
- La méthode la plus efficace de transmettre l'information est une conversation en face à face.

- L'unité de mesure de la progression du projet est un logiciel fonctionnel (ce qui exclut de comptabiliser les fonctions non formellement achevées).
- Les processus agiles promeuvent un rythme de développement soutenable (afin d'éviter la non qualité découlant de la fatigue).

- Les processus agiles recommandent une attention continue à l'excellence technique et à la qualité de la conception.

- La simplicité et l'art de minimiser les tâches parasites, sont appliqués comme principes essentiels.

- Les équipes s'auto-organisent afin de faire émerger les meilleures architectures, spécifications et conceptions.

- À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son processus de travail en conséquence.

Le but de l'outil

L'objectif général, exigé en programmation d'algorithmes distribués (PAD), est de développer une application distribuée sur l'environnement JAVA/J2EE en utilisant un maximum de technologies afin de se familiariser avec cet environnement. Le choix du sujet est libre mais encadré et doit être validé.

L'outil choisi, que nous développerons, doit répondre à un besoin existant, aujourd'hui, en gestion de projets informatique :

« Conduire un projet en appliquant les valeurs communes et les principes fondamentaux des méthodes agiles »

Contrairement à certaines applications de conduite de projets actuelles, l'outil à développer se devra d'être un moyen d'aide à la prise de décision en facilitant l'organisation du travail, l'autogestion de l'équipe de développement et le travail du chef de projet (et product-owner). En aucun cas, l'outil doit servir de justification à une décision puisque cela serait contraire aux valeurs et principes de l'agilité.

Pourquoi ce choix

L'idée est venue de l'un des membres du groupe de travail, lors d'un entretien téléphonique pour la recherche de stage. Sa correspondante, chargée d'affaires dans une SSII (Société de Services en Ingénierie Informatique), lui présente quelques logiciels internes à l'entreprise et lui expose une application de gestion de projets agiles car « *ces solutions sont quasi inexistantes sur le marché* ». C'est alors que le projet né. Des recherches sont faites pour voir l'état de l'art et effectivement : il n'existe aucun logiciel de conduite de projets spécifiques aux méthodes agiles. Il existe une multitude de gestionnaire de projets mais un seul propose l'agilité en « plug-in » : Jira.

Le projet est alors validé par le groupe de travail pour plusieurs raisons : il nous permettra l'utilisation d'un maximum de technologies et son application est réellement professionnelle. Ainsi, il sera, pour chacun du groupe, un très bon exemple d'utilisation du framework J2EE lors de nos entretiens d'embauche.

Dans la suite du document, on appellera « *outil idéal* » l'application développée, finalisée et pouvant être déployée dans n'importe quelle structure professionnelle. Cet outil étant complexe et son nombre de fonctionnalités étant relativement conséquent, le cahier des charges suivant présentera essentiellement l'objectif visé dans le cadre du projet « PAD » à savoir un sous-ensemble des fonctions de l'outil idéal.

CAHIER DES CHARGES

Nous allons, dans un premier temps, étudier les fonctionnalités choisies, à développer, et pour chacune, les choix pris en matière de technologies de développement. Enfin, nous finirons en détaillant toute la partie analyse et conception du futur outil en mettant en évidence son architecture EJB (Enterprise JavaBeans).

L'outil à développer et les choix technologiques

Bien qu'ayant suivi des cours de gestion de projets agiles, nos connaissances restent basiques face à l'immensité du domaine. C'est pourquoi, nous avons demandé l'aide de notre professeur : M. Fabien Peureux qui connaît parfaitement la notion d'agilité. Nous avons donc assisté à plusieurs réunions avec lui en partant du principe qu'il était notre client : « *Quels sont vos besoins ? Quelles fonctionnalités, que les outils actuels ne possèdent pas, souhaiteriez-vous ? ...etc.* »

Après quelques réunions et une étude approfondie des outils existants, nous avons listé toutes les fonctionnalités que l'outil idéal doit présenter. Nous en avons choisi un sous-ensemble pour le projet « *PAD* ». Ce choix a été motivé par des critères de préférences par chacun des membres du groupe et par la vision, commune au groupe, d'approfondir nos compétences avec un maximum de technologies. Ainsi, une fonctionnalité qui nous ferait utiliser la même technologie qu'une précédente ne serait pas retenue.

L'interface graphique

L'interface graphique a été l'une des étapes les plus longues car elle se doit d'être simple, conviviale mais à la fois efficace et professionnelle pour répondre aux besoins des futurs utilisateurs. Nous avons, pour cela, étudier les différents logiciels de gestion de projets (l'agilité ici n'étant pas un critère décisif) et rechercher des solutions efficaces pour arriver au choix d'une interface en « *widgets* ».

Les « *widgets* » permettent, en effet, de moduler l'interface graphique selon les besoins de l'utilisateur et notamment de son rôle au sein d'un projet. Ainsi, un chef de projet n'aura, peut-être, pas la même mise en page qu'un développeur mais le noyau des deux interfaces sera identique. Cela facilite le développement du rendu de l'application tout en répondant aux besoins de personnalisation selon les profils.

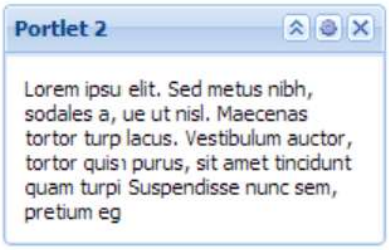


Figure 1 : Exemple d'un widget

De nombreuses technologies existent pour le développement de widgets, c'est pourquoi, une étude a été réalisée par trois des membres du groupe afin de mettre en évidence les avantages et inconvénients de chacune et permettre à l'ensemble du groupe de prendre une décision quant à la technologie de développement.

Le tableau suivant est le résultat de cette étude.

Figure 2 : Comparaison des technologies pour l'IHM

Technologies	Avantages	Inconvénients
HTML/CSS JavaScript (ExtJS)	- Gratuit - Documentation abondante - 2 membres initiés - Grande palette de composants - Adaptation mobile possible	- Pas d'IDE
HTML/CSS JavaScript (Jquery)	- Gratuit dans le cadre de projets libres - Documentation abondante - 3 membres initiés - Grande palette de composants - Adaptation mobile possible	- Pas d'IDE
Adobe Flex	- SDK gratuit - Documentation importante - Facilité d'interactions avec les WebServices - Simplicité	- IDE existant mais payant - Nécessite un "plug-in" pour les navigateurs - Adaptation mobile difficile - Aucun membre initié
JavaServer Faces PrimeFaces	- Gratuit - Repose sur Java/Eclipse donc tous initiés - Documentation importante - Adaptation mobile possible	
Google Web Toolkit	- Gratuit - Repose sur Java/HTML donc facilité - Adaptation mobile possible	- Peu de documentation - Aucun initié
HTML/CSS Google Dart	- Gratuit - Simplicité de JavaScript - Adaptation mobile possible	- Pas de documentation ou très peu - Aucun initié
Oracle ADF	- Gratuit - Java Code Portable (web/mobile)	- Peu de documentation - Aucun initié

Toutes les technologies, citées lors de cette étude, répondent aux besoins de développement de widgets. Toutes, pouvaient nous apporter de nombreuses connaissances mais le choix s'est fait par rapport à certains critères. Nous voulions une technologie nouvelle mais connue par certains membres du groupe pour ne pas perdre trop de temps, ainsi qu'une documentation et une communauté assez riches afin de trouver des solutions rapides à des problèmes. Enfin, nous voulions quelque chose de gratuit qui puisse être intégrée rapidement (pas de « *plug-in* » additionnel).

C'est pourquoi, le choix s'est porté sur la combinaison de HTML/CSS pour la structure générale des pages et de JavaScript notamment sa bibliothèque « *ExtJS* » permettant de construire des pages interactives : de nombreux exemples existants, une documentation et une communauté abondantes, quelques membres du groupe déjà initiés à JavaScript mais aucun à « *ExtJS* ».

Ci-contre, un exemple d'interfaces graphiques possibles en JavaScript.

Figure 3 : Exemple d'interface en widgets

Une fois, la technologie choisie et les possibilités d'IHM étudiées, nous nous sommes concentrés sur l'architecture de l'interface. Le schéma suivant est une maquette d'une vue « utilisateur ». Les vues seront personnalisées selon le rôle de l'utilisateur dans le projet : chef de projet, développeur…

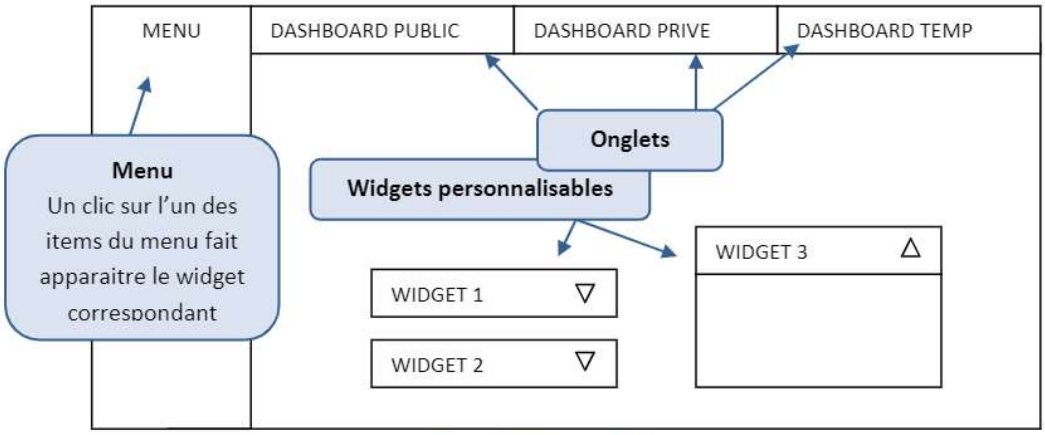


Figure 4 : Une vue utilisateur

Les « Dashboard » sont des espaces de travail disponibles pour chaque utilisateur, quelque soit son rôle. Il en existera trois : public, privé et temporaire comme détaillé ci-dessous. Le menu est général et le clic sur un item du menu fera apparaitre immédiatement le widget correspondant dans le Dashboard ouvert (ou qui a le focus).

- Le Dashboard public est défini par le chef de projet (ou le product-owner) afin de mettre en évidence certains widgets essentiels aux développeurs (la progression du projet, les bugs éventuels…). Ces derniers n'ont pas les droits pour le modifier. Seul le chef de projet peut modifier cette vue. Les autres utilisateurs sont juste lecteurs.

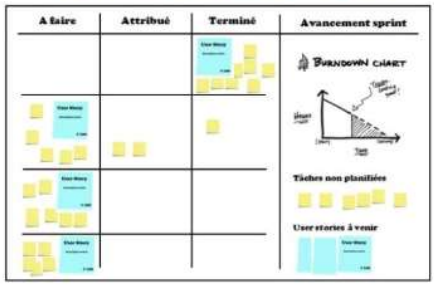
- Le second Dashboard, privé cette fois-ci, est propre à chaque utilisateur. Ainsi, chacun peut définir ces propres widgets. L'état de ce Dashboard est sauvegardé à chaque fin de session et chargé à chaque réouverture.

- Enfin, le dernier est considéré comme un espace de travail temporaire. A chaque fermeture de session, cet espace est supprimé.

Le « TashBoard » *(ou le tableau de tâches)*

C'est la principale fonctionnalité qui a donné lieu, d'ailleurs, au projet. En effet, les entreprises utilisant les méthodes agiles ont, pour la plupart, un tableau accroché au mur avec pour chaque tâche : une fiche. Ce tableau est composé de trois colonnes représentant un statut : tâches à faire, tâches en cours et, enfin, tâches réalisés ; comme le montre l'illustration ci-contre.

Figure 5 : Exemple de « TashBoard »



Le but est de développer un « *TashBoard* » numérique où il est facile de créer/supprimer une fiche et de la glisser/déposer dans l'une des colonnes du tableau. Ce TashBoard sera un item du menu et sera donc accessible via un widget spécifique.

Pour réaliser ceci, nous nous sommes concentrés sur deux types de développement : le premier est la solution Flex, propriété d'Adobe, permettant, en autre le « *drag & drop* » et facilitant le travail collaboratif. Le second est la combinaison du framework JSF (Java Server Faces) et de la bibliothèque de composants PrimeFaces.

Le tableau ci-dessous présente les avantages et inconvénients de ces technologies.

Technologies	Avantages	Inconvénients
Adobe Flex	- SDK gratuit - Documentation importante - Facilité d'interactions avec les WebServices - Simplicité - Possibilité de collaboration	- IDE existant mais payant - Nécessite un "plug-in" pour les navigateurs et donc adaptation mobile difficile - Aucun membre initié
JavaServer Faces PrimeFaces	- Gratuit - Repose sur Java/Eclipse donc tous initiés - Technologie JAVA/J2EE - Documentation importante - Adaptation mobile possible	

Figure 6 : Comparatif des technologies pour le TashBoard

Au vu de la colonne « *inconvénients* », c'est la combinaison JSF/PrimeFaces qui a été retenue. Le choix a été motivé par le fait que ces technologies font fortement parties de l'environnement JAVA/J2EE donc adéquates au projet et qu'il y a une documentation suffisante dessus.

Nous nous étions penchés, au départ, sur Adobe Flex car nous avons cru que le tableau de tâches (TashBoard) se devait d'être collaboratif. Après demande auprès de M. Peureux, le tableau n'est pas collaboratif mais est le résultat d'une collaboration directe entre chaque membre d'une équipe de développement.

La gestion des utilisateurs et de leurs droits

Comme cité précédemment, l'environnement graphique de notre application doit s'adapter aux profils des utilisateurs. Un développeur n'aura donc pas les mêmes possibilités d'actions sur l'ensemble du projet qu'un product-owner ou qu'un chef de projet.

Par conséquent, une partie du travail consistera à donner la possibilité à l'administrateur de l'application de pouvoir créer des utilisateurs, des groupes d'utilisateurs et de leur affecter des droits spécifiques à chacun.

Le schéma suivant explique cette procédure.

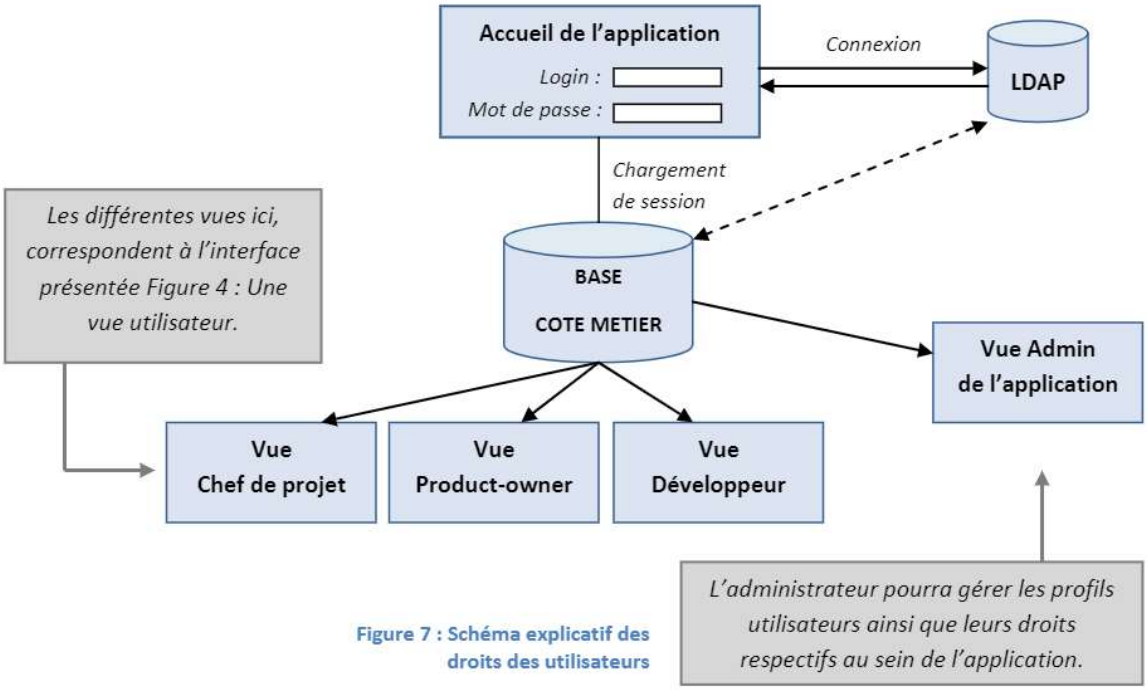


Figure 7 : Schéma explicatif des droits des utilisateurs

La base de données de notre application a deux fonctions : elle regroupe toutes les données de gestion de projets (tâches, itérations, caractéristiques du projet…) et elle sauvegarde les vues des utilisateurs chargées à l'ouverture de leur session.

Lors de sa connexion, un utilisateur se voit chargé sa session : les menus lui correspondants (par rapport à ses droits) et ses Dashboards : le public définit par le chef de projet, le privé (propre à l'utilisateur) restauré dans l'état qu'il était lors de la dernière déconnexion et enfin le temporaire vide de tout widget.

Dans cette partie, nous avons décidé d'utiliser l'API Java JNDI pour la connexion à l'annuaire LDAP. Ce choix a été tout normal car l'ayant vu en cours, nous voulions s'améliorer dessus.

Un second choix a été nécessaire pour la base de données et la connexion vers cette dernière. Nous partons, sur l'API JPA et une base MySQL. Nous avons remarqué qu'Hibernate était fortement utilisé dans le monde professionnel mais nous souhaitons utiliser JPA car c'est une API standard et interchangeable : tout développement sous JPA fonctionne parfaitement sous Hibernate, TopLink ou encore OpenJPA. De plus, nous sommes partis sur une base de données MySQL car la majorité du groupe connait cette base : cela évite tout problème lors du développement.

Voici, les trois grandes étapes de développement qui fixent l'objectif de notre projet. Bien entendu, nous avons volontairement présenté ce cahier des charges en faisant abstraction des petites tâches induites par ces grandes phases.

Nous avons ici une idée générale de l'application à développer : un outil où l'on se connecte, où l'on accède à un environnement spécifique suivant notre rôle et où l'on peut gérer des tâches grâce à un tableau.

La seconde partie de ce document présente les phases d'analyse et de conception de l'outil.

Analyse et conception

Pas de grandes phrases pour présenter cette partie. Nous nous focaliserons directement sur les schémas et diagrammes en les détaillants.

Architecture EJB de l'application

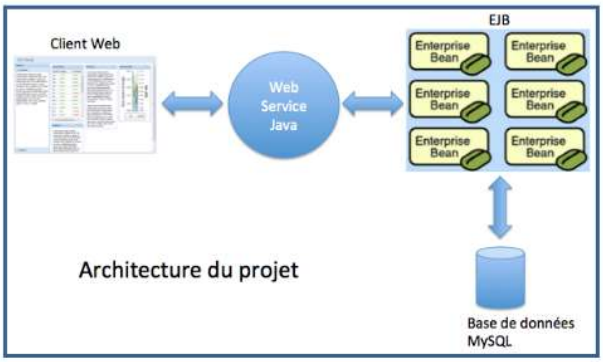
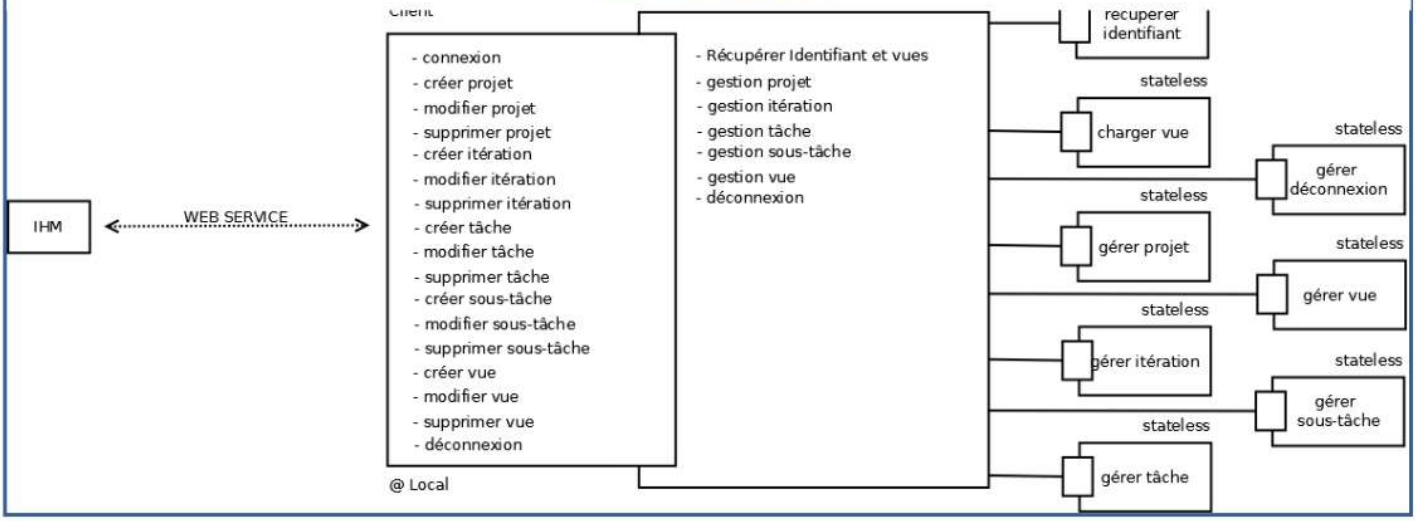


Figure 8 : Architecture générale de l'application

Concernant l'architecture globale du projet, nous avons choisi de la décomposée en trois couches : présentation, métier et accès au données.

Un service web Java effectue l'interface entre le client web et les parties métier/données. Afin d'accéder au web services, nous avons choisi d'utiliser la technologie AJAX (*Asynchronous Javascript and XML*) afin de faire des appels serveur en mode asynchrone.

Figure 9 : Architecture EJB



La fonction « *connexion* » permet de faire le lien entre LDAP et la base de données MySQL.

Dans l'annuaire LDAP, on obtient l'identifiant de l'utilisateur afin de le retrouver dans la base de données. La fonction « *connexion* » retourne également un numéro de session au client. On peut alors charger les vues via la base de données.

Deux EJB de type « *stateless* » seront utilisés : le premier permet de récupérer l'identifiant et le deuxième permet de charger les vues.

Pour la gestion du projet, il est possible de créer un projet, de le modifier et de le supprimer. Pour ce faire, un EJB de type « *stateless* » est utilisé. Le même principe est utilisé pour la gestion des itérations, des tâches et des sous-tâches. Avant d'effectuer le traitement demandé, on vérifiera que le client possède les droits pour réaliser ce type de demande. Par exemple, le product-owner a le droit de créer, modifier et supprimer une tâche contrairement aux développeurs.

En ce qui concerne la gestion des vues, les vues personnelles des utilisateurs seront enregistrées temporairement. Lors de la déconnexion de l'utilisateur, un EJB de type « *stateless* » enregistrera la vue personnelle de l'utilisateur dans la base MySQL. De ce fait, cette vue pourra être chargée lors de la prochaine connexion de l'utilisateur.

Les pages suivantes présentent respectivement le diagramme de classes et le schéma de la base de données MySQL. Bien entendu, les attributs et fonctions pourront évoluer au fil du développement.

Diagramme UML de classes

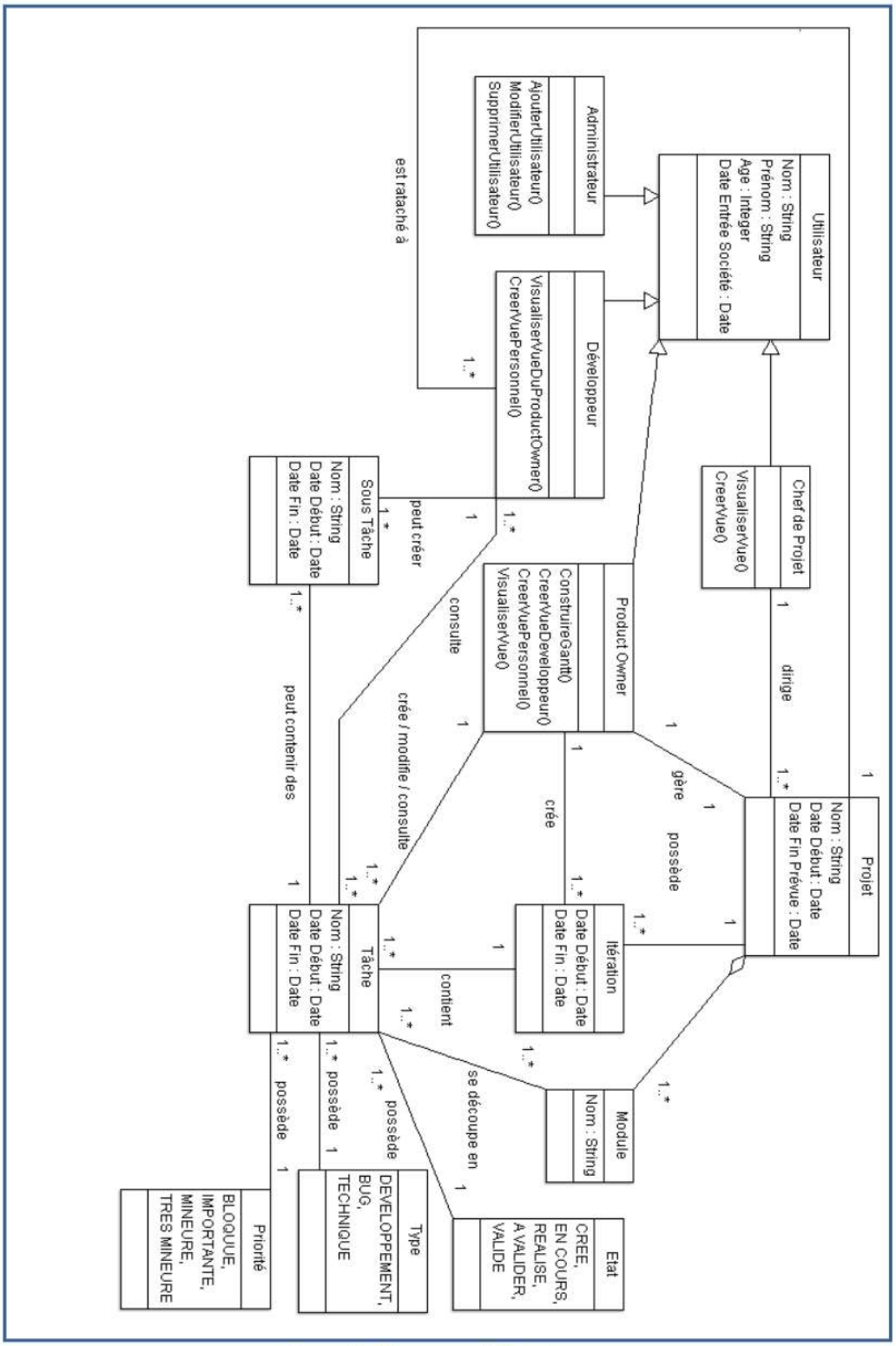


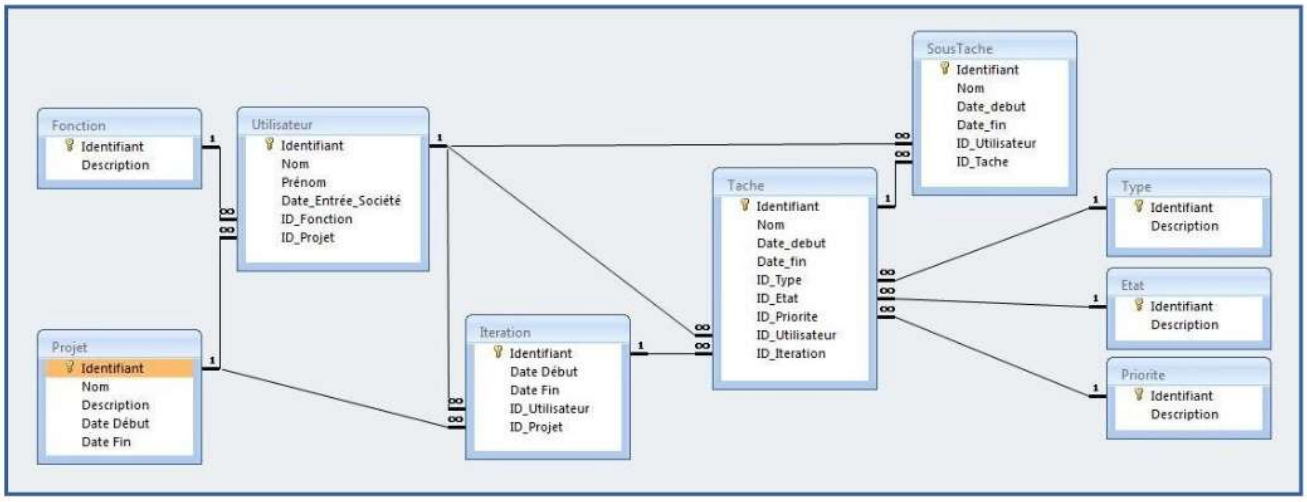
Figure 10 : Diagramme UML de classes

Schéma de la base de données

Comme expliqué précédemment, la base de données présente deux cotés : tout ce qui concerne le chargement des vues (position des widgets, quels widgets…) et le coté « *métier* » en rapport aux données de gestion de projets (tâches, itérations…).

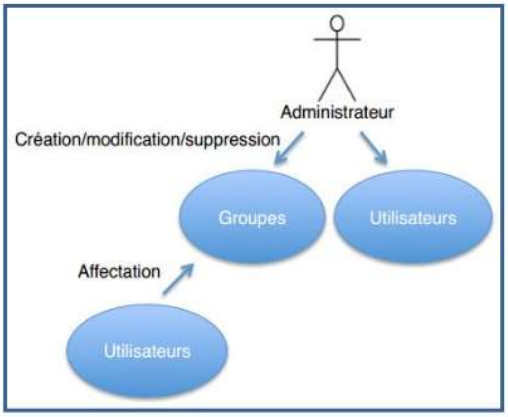
Dans le schéma, présenté ci-dessous, nous détaillons juste la partie « *métier/données* » en faisant, volontairement, abstraction de la partie *« chargement des vues »*. En effet, cette dernière sera précisée au cours du développement car on ne sait pas réellement les attributs et fonctions.

Figure 11 : Schéma de la base de données



Use-cases

Afin de donner une vision globale du comportement de l'application, voici trois diagrammes de cas d'utilisation correspondant aux différents types d'utilisateurs : l'administrateur de l'application, le product-owner (chef de projet) et enfin le développeur.



L'administrateur de l'application a la charge de gérer les utilisateurs ainsi que les groupes d'utilisateurs.

Figure 12 : Use-case Administrateur

Concernant les fonctions du product-owner, celui ci à la charge de gérer son projet. Dans un premier temps, celui ci doit affecter des développeurs à son projet. Ensuite, il pourra décomposer le projet en module et pour chaque module, créer une liste de taches. Enfin, il pourra planifier les itérations du projet correspondant à un ensemble de taches à réaliser dans un délai imparti.

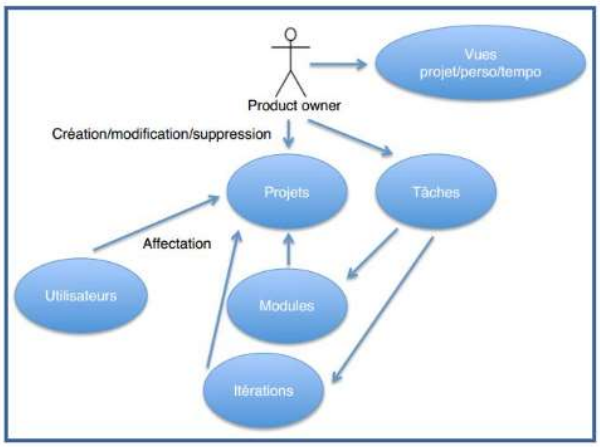


Figure 13 : Use-case « Product-owner »

Il dispose également du droit de proposer une vue (ensemble de widgets) à l'ensemble des développeurs du projet.

Un développeur, en partenariat avec d'autres développeurs s'affecte des taches à réaliser lors d'une itération. Il dispose également le droit de créer des sous taches à une tache créée par le chef de projet. Une foi terminée, il dispose de la possibilité de modifier son état (en cours, réalisée, à valider, validée).

Concernant les vues, il se voit affecter la vue par défaut du chef de projet. Il peut également créer ses propres vues dans le Dashboard personnel et temporaire.

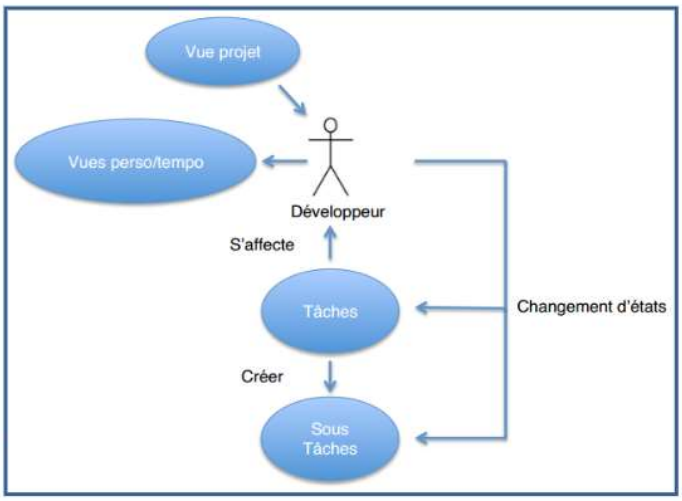


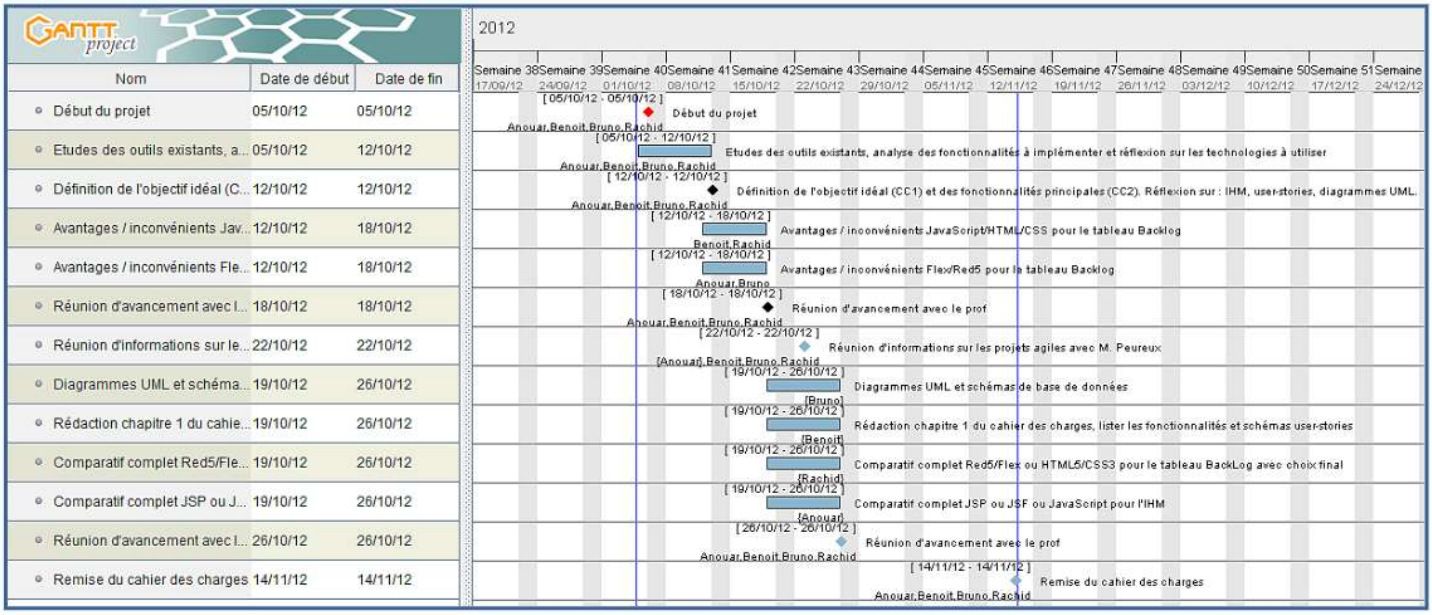
Figure 14 : Use-case Développeur

Gestion du travail et des outils

Pour réaliser ce projet de gestion de projets agiles, nous-mêmes n'utiliserons pas l'agilité car elle implique un travail en binôme et étant un nombre impair cela compliquerait la chose. Néanmoins, nous allons reprendre certaines notions de l'agilité. Ainsi, nous nous réunirons chaque semaine, au moins une fois, pour voir l'avancer de chacun. Nous faciliterons l'autogestion de l'équipe en favorisant la communication directe et le choix, pour chaque membre, de prendre telle ou telle tâche de développement.

Aucun chef de projet n'est désigné mais une personne assurera le maintien et le suivi de la gestion du projet en tenant à jour un diagramme de GANTT.

Ci-dessous, un aperçu du diagramme GANTT jusqu'à la remise de ce rapport.



En plus de ces méthodes organisationnelles, nous utiliserons des outils spécifiques. Le développement se fera en JAVA sous l'environnement Eclipse avec les technologies présentées précédemment et grâce à un serveur Jonas.

Un dépôt SVN a déjà été créé afin de développer en équipe.

GESTION DE PROJETS AGILES

EQUIPE DE DEVELOPPEMENT

Benoit Bernardin

Rachid Bouherrou

Bruno Juillard

Kaba Kabinè

Sid-ahmed Anouar Lekhal