POLYTECH MONTPELLIER

PolyDiyManager

Loïc DUMAS
Pierre CASATI
Faustine GEOFFRAY
Nassim VACHOR

Abstract

This paper lays out about our work and our contribution in this software engineering project. this project consists to design an application in java which help people to create their projetcs by themselves.
In fact, after a constitution of subgroups, we worked in first, on the conception of our application's class diagramme, and after we have designed all use cases and we have separed them on optionnal and requeried use cases. finally we began the development and the design of the requeried use cases.

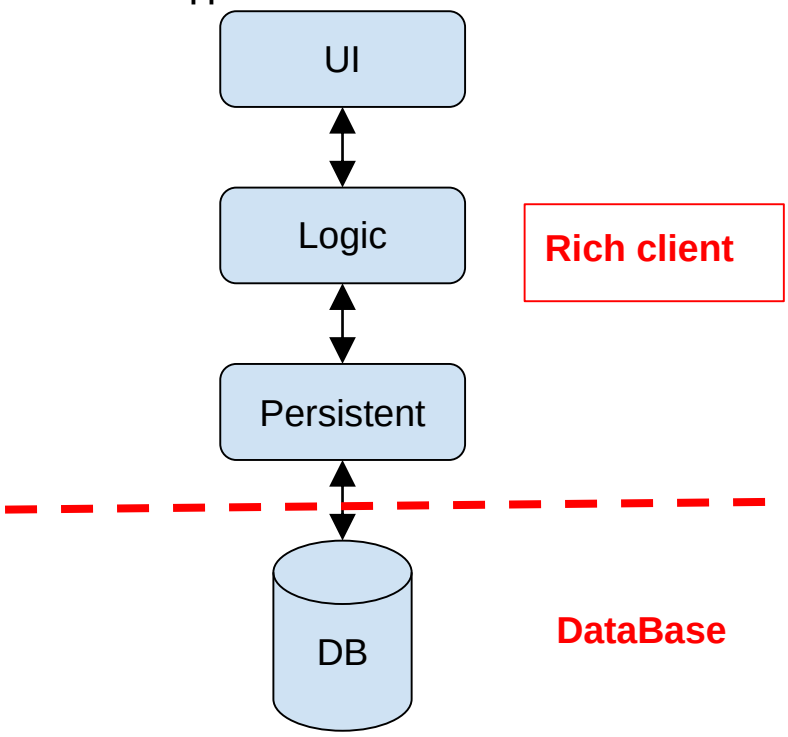
keys words: Java, Swing, design pattern, database , postgresql, GitHub, UML, Merise

Introduction

Dans le cadre de notre formation, nous avons eu comme devoir le développement d'une application destinée aux personnes voulant créer leur propres projet ou activité.
Ce projet a pour principal but, la mise en oeuvre des compétences acquises en java au premier semestre ainsi que les différentes connaissances acquises durant ces deux dernières années notamment en base de données.
Après la constitution des différents groupes, nous avons commencé dans un premier par une étude générale du travail à faire, ensuite nous avons désigné un manageur général de notre projet ainsi que les différents responsables de chaque étape de notre projet.
Ensuite, nous avons procédé à la modélisation de notre application en modélisant le diagramme de classe général de notre application. En outre, nous nous sommes penchés sur le design des différents use cases. Une fois cette partie est finie, nous avons procédé à la conception des différents UML détaillés de chaque use case.
Enfin, nous avons conclu notre projet par le développement des différents uses cases et des différents tests unitaires relatifs à chaque méthode.
Ce rapport a pour but l'exposition des différentes étapes de notre projet, les différentes difficultés rencontrées ainsi que les différents apports de ce projet.
Pour cela, il parait opportun de commencer ce rapport par la partie technique en abordant l'architecture utilisée, la conception et le développe. Ensuite, nous élaborerons la partie gestion de projet en abordant l'organisation de notre groupe, synchronisation du travail et le déroulement du projet.
Enfin, nous vous présenterons nos appréciations générales de ce projet ainsi que le point de vue de chaque membre.

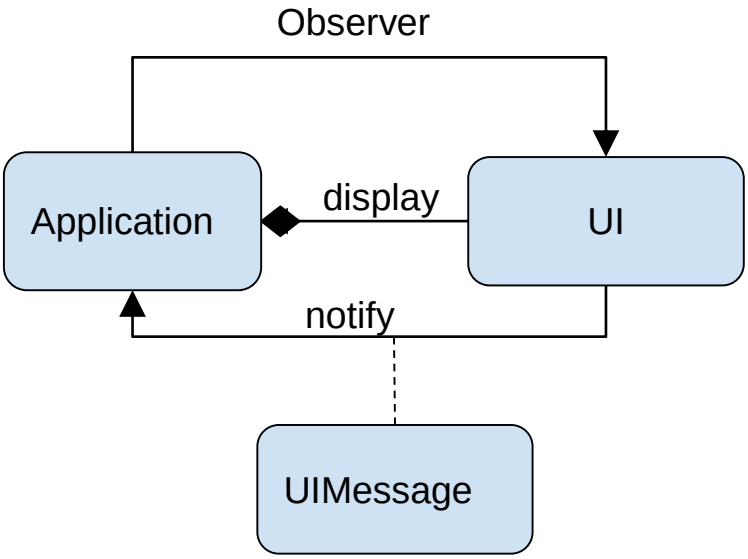
1. Technique

a. Architecture de l'application

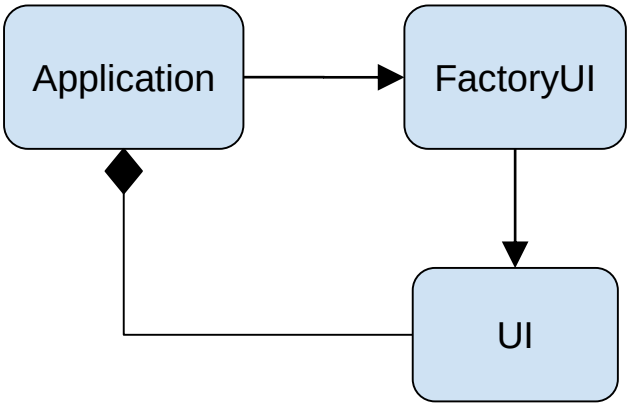


L'ensemble des services proposés reposes sur une architecture 2-tiers constituée d'un client lourd et d'un serveur de données.
Le serveur de données fonctionne sous postgreSQL et est déployé chez Heroku.
En ce qui concerne le client lourd, il est lui-même construit sur une architecture 3-tiers constituée de : la couche "UI" chargée de l'affichage, la couche "logic" chargée de la logique de l'application, et la couche "Persistent" chargée de la manipulation des données.

Fonctionnement de la couche UI :
La couche UI du client riche est chargée de la génération et de l'affichage des interfaces utilisateurs. C'est également cette couche qui gère les interactions avec l'utilisateur (remplissage de champs, clics).

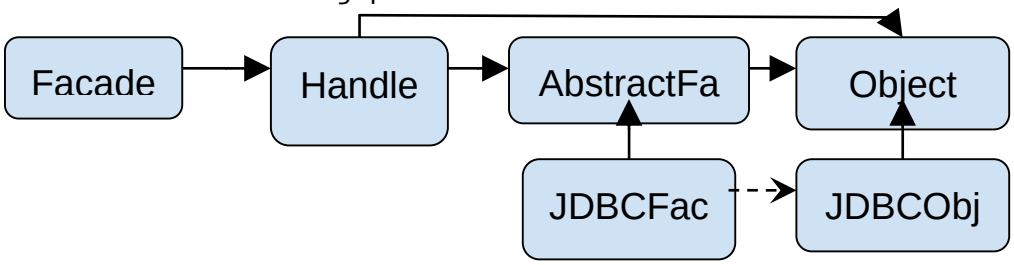


L'application est une fenêtre qui se remplit avec différentes UI. De plus, elle observe ces UIs afin de savoir quand nettoyer la fenêtre et quand afficher de nouvelles UI.
De plus, afin de pouvoir transmettre des informations entre les UI, lorsque celles-ci notifient l'application, elles lui envoient un UIMessage contenant les diverses informations, et cet UIMessage sera ensuite transmis aux nouvelles UIs.



Afin d'effectuer les changements d'UIs, l'application utilise une fabrique qui aura au préalable enregistré les méthodes de constructions de chaque UI et les aura associées à une clé d'indexation. Ainsi on pourra facilement retrouver et construire une UI en utilisant juste la clé associée.

Fonctionnement de la couche logique :



La couche logique de PolyDIY Manager est composée de toutes les classes handler. Ces classes sont toutes la partie logique de l'application.
Cette couche est reliée avec l'interface par une façade (package : common.facade)

permettant de rendre transparentes toutes demandes de l'UI à la partie métier. La façade se contente simplement de faire l'intermédiaire à l'UI pour la demande des ressources. L'UI n'aura plus la contrainte de devoir connaitre la partie logique, et s'y adapter s'il y a des changements, elle se contentera d'utiliser la façade, qui elle saura où aller chercher les bonnes ressources.

Les classes correspondantes se situent dans le package "Logic".

Les handler correspondent à la partie métier de chaque use case. LoginChecker par exemple, servira, lors de la connexion, à vérifier si l'utilisateur peut se connecter ou non. Il contient deux méthodes :

- `public boolean isValidPassword(String password);`
- `public void generateAccount(String login);`

IsValiPassword, prend un password et le compare (suite à un cryptage en sha256) au mot de passe présent en base de données. IsValidPassword renverra vrai si les mots de passe concordent, faux autrement. Ainsi IsValidPassword décidera de la création d'un objet Account ou non.

Dans le cas des Handler pour une wishList. Il appellera les factory pour créer les wishlist, et commandera les modifications de ces wishlist tout en s'assurant de la cohérence des données, vérifiera si ce produit n'existe pas déjà avant de demander sa création. Par exemple la méthode

- `public boolean addProductToWishList(int IDProduct, int quantity, float unitPrice)`

Vérifiera que la nouvelle quantité ne soit pas nulle ou négative, et pareil pour le prix unitaire et qu'il n'existe pas déjà dans cette liste. Une fois l'élément ajouté à la wishList, il sera inséré dans la base de données. Autrement des erreurs sont générées.

Fonctionnement de la couche persistante :

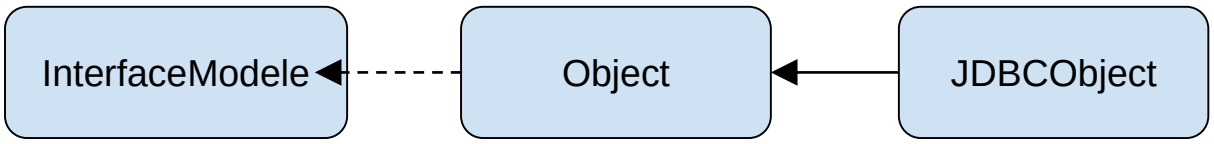
Afin de produire chacun de nos objets de la couche persistante nous utilisons différentes factory. Il y a :

- ActivityFactory : Pour créer les activités, tâches et objectives
- ListFactory : Pour créer tous les ensembles héritants de Set (WishList, Cart, ...)
- ProductFactory : Pour créer les produits et les item_wishList
- ProfileFactory : Pour créer les seller, les user (et les admin)
- SessionFactory : Pour créer les account et les session.

Les factory sont des classes abstraites qui sont ensuite étendues par les JDBCFactory, qui sont des factory spécifiques au modèle de données persistant choisi.
Ainsi lorsque l'on utilisera cette abstract factory, on voudra par exemple un produit, et simplement en appelant la méthode build correspondante, on obtiendra un JDBCProduct.

Pour le modèle, chaque objet est une classe abstraite contenant les informations nécessaires. (par exemple pour un produit, son nom, son prix unitaire et sa quantité) et sera ensuite étendue par une version JDBC de cet objet .

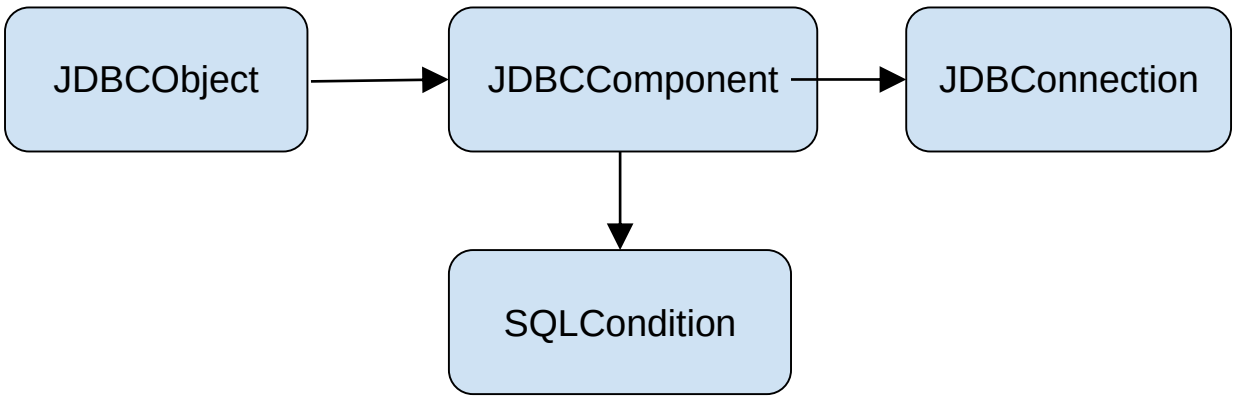
Le JDBCProduct contiendra lui les méthodes nécessaires à sa construction (en allant chercher dans la base de données) . Ill contient également les méthodes lui permettant de s'update, insert etc. (obtenu en implémentant l'interface modele)



D'autre part, afin de structurer notre méthode de travail, et pour simplifier le travail collaboratif, l'interface InterfaceModele a été déclarée. Cette interface définit un certain nombre de méthodes communes aux différents modèles et qui correspondent en fait au CRUD (Create, Read, Update, Delete). Ainsi un modèle définit uniquement la structure de données, et la version étendue JDBCModèle redéfinit les méthodes du CRUD en utilisant le package JDBC.

Il existe un autre type de modèle le SetWithKey<String,Object>. Il s'agit d'une hasmap. Cette classe a été utilisée à de nombreuses reprises pour tous les set, où l'on avait besoin de trouver le produit avec une Key. Cette structure s'adaptait donc très bien dans de nombreux cas à notre application. Par exemple les wishlist qui contiennent des products, ils sont donc référencés par leurs ID, permettant ainsi de bénéficier des méthodes du Set (addElement, ...)

Connexion à la Base de données :



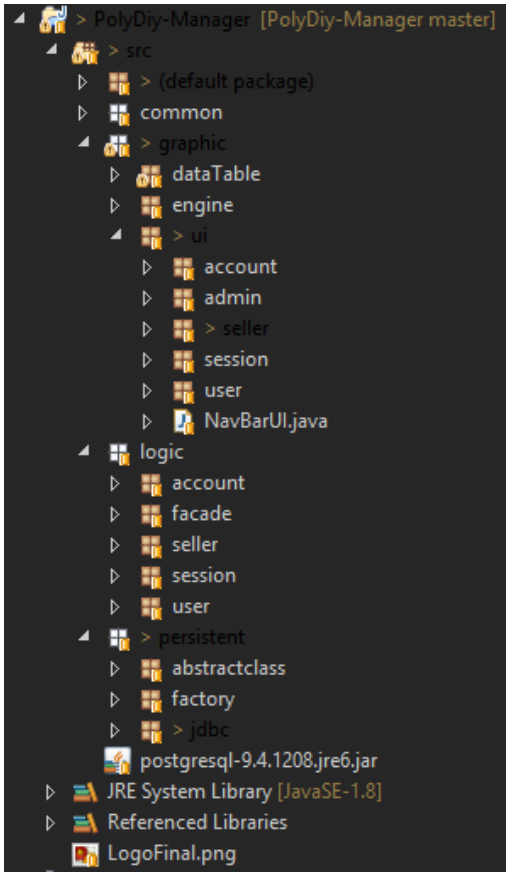
Afin de simplifier l'utilisation du package JDBC, et dans le but de factoriser le code lié aux requêtes, une surcouche a été créée pour JDBC.
Dans un premier temps, JDBConnection génère la connexion vers la base de données (en utilisant les identifiants de connexion), et en suivant le pattern singleton.
La classe JDBCComponent encapsule les requêtes SQL Select, Update, Insert, et Delete dans des méthodes simplifiées. Elle utilise également un objet SQLCondition pour simplifier l'écriture de la clause WHERE de certaines requêtes.
Ainsi, un objet utilisant JDBC peut être simplifié en passant par JDBCComponent.

Base de données :
La base de données utilise postgreSQL et est hébergée chez Heroku.

Nous avons préalablement réalisé un MCD, que nous avons ensuite dérivée en MLD, ce qui nous a servis à générer les scripts SQL de création de tables, puis d'insertion de tuples dans les tables.
Les MCD et MLD sont disponibles en annexe.

Les exceptions :
Afin de gérer les nombreux cas d'exception possible. PolyDIY Manager utilise de nombreuses exceptions disponibles dans le package common.exception.
Par exemple, lorsqu'un utilisateur tente de se connecter, si le mot de passe est invalide, une exception est renvoyée lui informant qu'il y a une erreur sur le mot de passe. Il en va de même lorsque l'on tente de rajouter un produit à une liste. Si dans la quantité on met une lettre par exemple, une erreur est affichée dans une boite de dialogue, informant l'utilisateur de son erreur.
Pour chaque exception, il y a deux possibilités : soit c'est une exception à destination du développeur et elle sera attrapée, soit c'est une erreur à destination de l'utilisateur, et elle sera affichée dans une boite de dialogue au niveau de l'UI concernée.

Les packages :
Afin de bien nous organiser lors de nos séances de programmations, nous avons choisi de respecter une hiérarchie de package. Nous avons séparé notre application suivant l'architecture en trois packages : "graphic", "logic", "persistent". Nous avons également un package par défaut contenant la class Main, et la base de l'application, et un package common, contenant des éléments utilitaires, par exemple la surcouche JDBC, les sets, ou les exceptions.



UML Global :

C'est UML (consultable en annexe) nous a permis de représenter les différentes fonctionnalités de notre application, et en particulier, les interactions entre les classes de la couche persistente.

2. gestion de projet

a. Notre plan

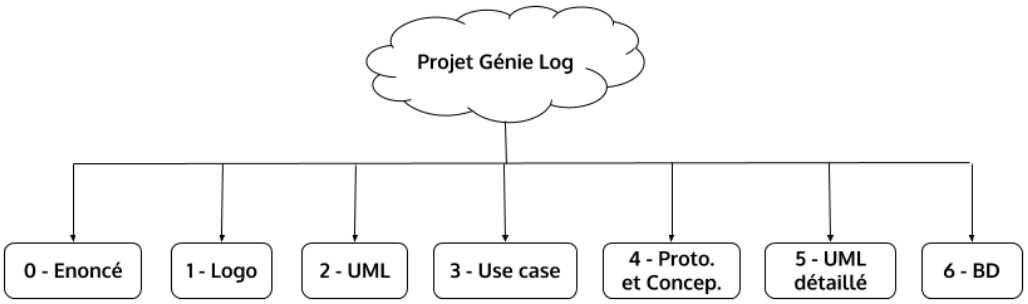
Ce projet s'est déroulé sur une période de deux mois. Afin de pouvoir tenir ce délai tout en atteignant les objectifs fixés, nous avions établi un planning prévisionnel.

date	n°étape	nom étape	finalité
1 fevr.	1	Formation du groupe	Définition des outils collaboratifs
12 fevr.	2	Modélisation et analyse	UML général
22 fevr.	3	Use case / maquette d'écran	Détail de chaque use case
29 fevr.	4	Prototype / conception	Use case "login" entièrement fonctionnelle
7 mars	5	Conception	UML détaillé pour chaque use case
31 mars	6	Développement / tests	Développement individuel sur des uses cases différents

Pour chacune de ces dates, nous devions rendre un compte rendu de notre travail, ce qui nous obligeait donc à respecter ce planning.

b. Synchronisation du travail

Dès la première étape de notre projet, soit "la formation du groupe", nous nous sommes mis d'accord sur les outils collaboratifs à utiliser tout au long de notre projet.
Ainsi, pour les premières étapes du projet, nous avons synchronisé notre travail grâce à Google Drive. Nous avons choisi d'organiser notre espace sur le drive, afin que chacun puisse instantanément retrouver les documents des autres. Ci-dessous un aperçu de l'arborescence de nos dossiers.



Dans chaque dossier se trouve un rapport de la partie, ainsi que quelques autres dossiers, souvent du type : "nom étape" + "nom du créateur des fichiers situés dans le dossier".
Afin qu'il n'y ait aucun problème d'extensions de fichiers, nous avions tous pris soin de télécharger "Visual Paradigm" pour réaliser nos UMLs.
En ce qui concerne la synchronisation de notre code, nos avons utilisé l'application Github. Pour ne faire aucune mauvaise manipulation avec cet outil, nous avons édité un "guide des bonnes pratiques", disponible à tout moment pour chaque membre de l'équipe. Ce guide restitue les principales commandes, dont certaines étaient encore inconnues pour une partie de l'équipe. Nous avons utilisé Winmerge pour résoudre les conflits de synchronisation avec simplicité.

Hormis les outils logiciels, nous organisions des réunions hebdomadaires, afin d'assurer un suivi régulier de nos avancées individuelles. Lors de ces réunions, nous présentions tous nos travaux ainsi que nos problèmes, afin de pouvoir discuter ensemble des améliorations à effectuer, et éventuellement afin de résoudre divers problèmes.

c. Séparation des tâches

Le projet était décomposé en 10 tâches qui pour chacune d'entre elle un responsable a été désigné. Ci-dessous le tableau vous présentant les responsables de chaque tâche.

Tâche	Responsable
Responsable Général	Loïc
Modélisation	Nassim
Use cases	Nassim
Design	Pierre
Développement	Faustine

Test	Loïc
JavaDoc	Pierre
Soutenance	Pierre
Base de données	Loïc
Design Graphique	Faustine
Connexion BD	Nassim

Comme le but de ce projet n'était pas seulement de rendre une application fonctionnelle, mais avant tout de réussir à s'organiser afin que chaque membre du groupe s'implique dans chacune des étapes, nous avions mis en place un roulement en fonction des familles de use cases à concevoir. Nous vous présentons ci-dessous le tableau utilisé pour savoir à tout moment qui avait travaillé sur telle partie du projet. La première colonne représente le nom des uses cases développées, la deuxième désigne la personne ayant rédigé le use case et fait les maquettes d'écran correspondantes. La colonne "UML détaillé" correspond à nos premiers UMLs détaillés, qui ont par la suite subit beaucoup de modifications, c'est pourquoi nous avons une deuxième version de ces umls. Lorsque nous avons réalisé que nous ne pourrions pas tout développer, nous avons instauré une priorité (1 étant très important et 3 moins important) à chaque use case. Enfin, la dernière colonne de ce tableau correspond à la personne qui a développé le use case.

	Use cases	UML détaillé	UML Version 2	Priorité	Dev
Structure du programme	Groupe	Groupe	Groupe	Done	Pierre
Login	Groupe	Groupe	Groupe	Done	Groupe
Create account	-	Pierre	Pierre	1	Pierre
Seller - customer interraction	Nassim	Loïc	Loïc	1	-
Seller - manage his shop	Nassim	Loïc	Loïc	1	Nassim
Seller - Describe Profil	Nassim	Loïc	Loïc	2	Nassim
Admin - manage account	Faustine	Pierre	Pierre	3	-
Admin - manage category	Faustine	Pierre	Pierre	2	-
Account - create tutorial	Pierre	-	-	-	-
Account - log off	Pierre	Pierre	Pierre	1	Pierre
Account - manage account	Pierre	Faustine	Faustine	2	Faustine
Account - "wall"	Pierre	Loïc	Pierre	1	*Pierre*
Account - notification	Pierre	-	-	-	-
User - Manage estimate	Loïc	Nassim	Faustine	1	-
User - Manage journal	Loïc	-	-	2	-
User - manage list	Loïc	Nassim	Faustine	1	Loïc
User - Request activity category	Loïc	Nassim	-	-	-
User - Shop - add to wishlist/cart	Loïc	Nassim	Nassim	2	Loïc

User - Activity Panel	Loïc	Faustine	Loïc	2	Pierre

d. Déroulement des étapes

Chaque étape a rencontré ses propres difficultés. Nous allons donc vous faire un résumé du déroulement de chacune d'entre elles.

La modélisation

Nous avons réalisé la modélisation en groupe lors de nos réunions hebdomadaires.

Les use cases

La définition des use cases et des familles de use cases s'est faite en groupe, ensuite chacun a écrit les use cases correspondant à une famille avec les maquettes d'écran correspondantes.

Le design

Pour réaliser cette étape, nous nous sommes beaucoup inspiré du use case "login" présentée en cours par M. Stratulat.

Le développement

Le niveau en développement très hétérogène au sein de notre groupe nous a été bénéfique. En effet, les personnes ayant déjà de l'expérience en développement Java se sont beaucoup impliquées au niveau de la gestion du projet. En revanche, les personnes n'ayant aucune expérience dans le domaine du développement en java ont appris énormément durant cette étape (fonctionnement Eclipse, bonnes pratiques...).

Les tests

Les tests ont été réalisés en groupe lors de la dernière réunion hebdomadaire.

La JavaDoc

Chaque personne du groupe devait écrire la documentation correspondant à son code. À cause de certains oublis, nous avons dû revenir à cette étape en fin de projet.

La Soutenance

Pour nous préparer à la présentation de notre projet, nous avons tous pris le temps de regarder entièrement le code de l'application, afin de pouvoir si nécessaire poser des questions.

La base de données

Nous avons développé notre base de données depuis PostgreeSQL. Nous avons aussi créé un petit script permettant d'insérer des données dans la base.

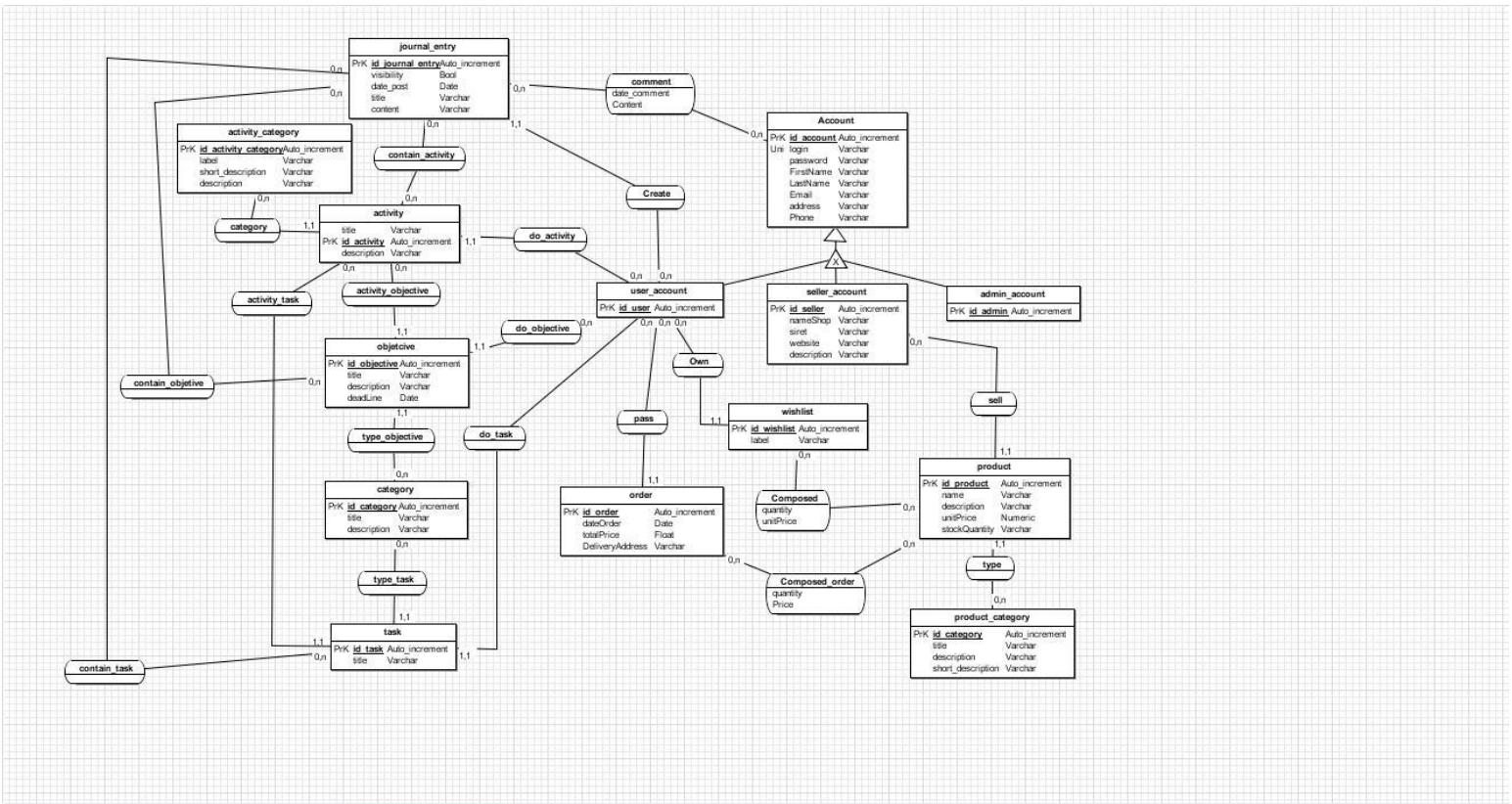
Design Graphique

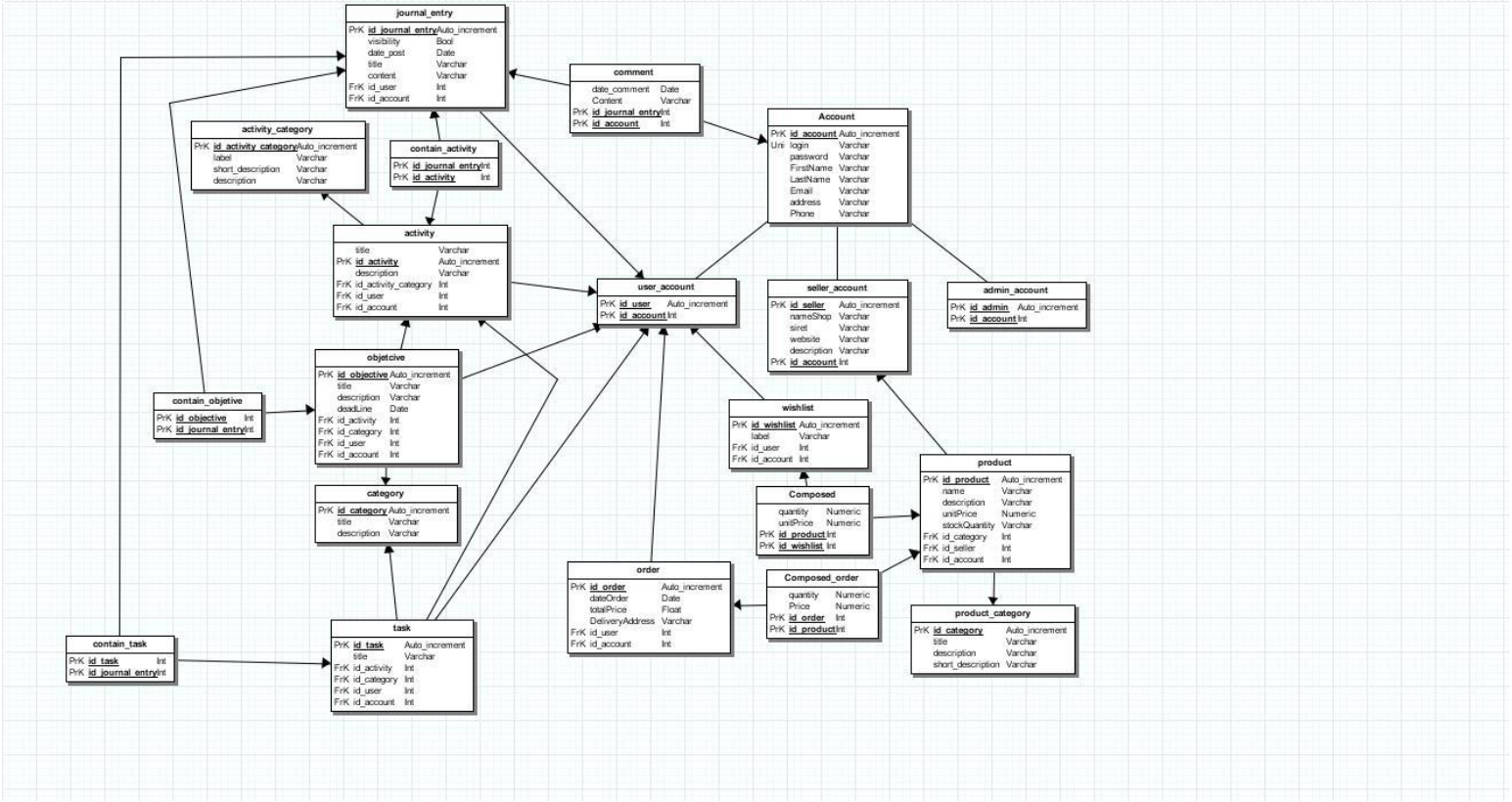
Pour le design graphique, chaque personne du groupe a développé ses propres UI. Pour accéder à chacune de ces UI, nous postions un bouton sur la page d'accueil.

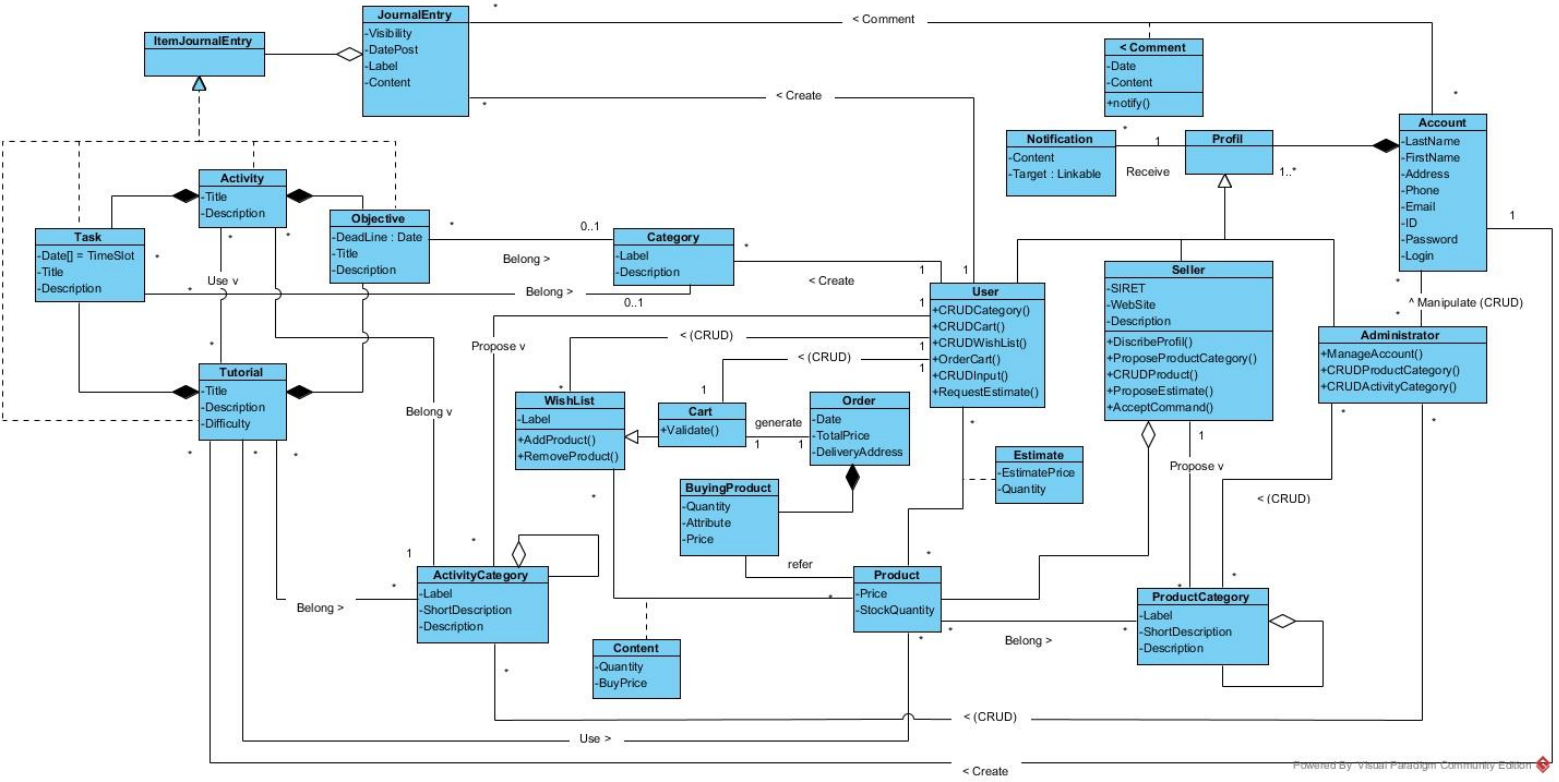
Connexion BD

Dans un premier temps nous avons développé notre BD en local. Afin de pouvoir travailler avec la BD sur plusieurs postes différents, nous avons décidé de l'héberger sur Heroku.

ANNEXES


journal_entry
PrK id_journal_entry Auto_increment
visibility Bool
date_post Date
title Varchar
content Varchar
comment
date_comment Date
Content
Account
PrK id_account Auto_increment
Uni login Varchar
password Varchar
FirstName Varchar
LastName Varchar
Email Varchar
address Varchar
Phone Varchar
activity_category
PrK id_activity_category Auto_increment
label Varchar
short_description Varchar
description Varchar
contain_activity
Create
category
activity
title Varchar
PrK id_activity Auto_increment
description Varchar
do_activity
activity_task
activity_objective
user_account
PrK id_user Auto_increment
seller_account
PrK id_seller Auto_increment
nameShop Varchar
siret Varchar
website Varchar
description Varchar
admin_account
PrK id_admin Auto_increment
objetcive
PrK id_objective Auto_increment
title Varchar
description Varchar
deadLine Date
do_objective
contain_objetive
Own
sell
pass
wishlist
PrK id_wishlist Auto_increment
label Varchar
type_objective
do_task
product
PrK id_product Auto_increment
name Varchar
description Varchar
unitPrice Numeric
stockQuantity Varchar
category
PrK id_category Auto_increment
title Varchar
description Varchar
order
PrK id_order Auto_increment
dateOrder Date
totalPrice Float
DeliveryAddress Varchar
Composed
quantity
unitPrice
type_task
Composed_order
quantity
Price
type
task
PrK id_task Auto_increment
title Varchar
product_category
PrK id_category Auto_increment
title Varchar
description Varchar
short_description Varchar
contain_task


journal_entry
PrK id_journal_entry Auto_increment
visibility Bool
date_post Date
title Varchar
content Varchar
FrK id_user Int
FrK id_account Int
comment
date_comment Date
Content Varchar
PrK id_journal_entry Int
PrK id_account Int
Account
PrK id_account Auto_increment
Uni login Varchar
password Varchar
FirstName Varchar
LastName Varchar
Email Varchar
address Varchar
Phone Varchar
activity_category
PrK id_activity_category Auto_increment
label Varchar
short_description Varchar
description Varchar
contain_activity
PrK id_journal_entry Int
PrK id_activity Int
activity
title Varchar
PrK id_activity Auto_increment
description Varchar
FrK id_activity_category Int
FrK id_user Int
FrK id_account Int
user_account
PrK id_user Auto_increment
PrK id_account Int
seller_account
PrK id_seller Auto_increment
nameShop Varchar
siret Varchar
website Varchar
description Varchar
PrK id_account Int
admin_account
PrK id_admin Auto_increment
PrK id_account Int
objetcive
PrK id_objective Auto_increment
title Varchar
description Varchar
deadLine Date
FrK id_activity Int
FrK id_category Int
FrK id_user Int
FrK id_account Int
contain_objetive
PrK id_objective Int
PrK id_journal_entry Int
wishlist
PrK id_wishlist Auto_increment
label Varchar
FrK id_user Int
FrK id_account Int
product
PrK id_product Auto_increment
name Varchar
description Varchar
unitPrice Numeric
stockQuantity Varchar
FrK id_category Int
FrK id_seller Int
FrK id_account Int
Composed
quantity Numeric
unitPrice Numeric
PrK id_product Int
PrK id_wishlist Int
category
PrK id_category Auto_increment
title Varchar
description Varchar
order
PrK id_order Auto_increment
dateOrder Date
totalPrice Float
DeliveryAddress Varchar
FrK id_user Int
FrK id_account Int
Composed_order
quantity Numeric
Price Numeric
PrK id_order Int
PrK id_product Int
product_category
PrK id_category Auto_increment
title Varchar
description Varchar
short_description Varchar
task
PrK id_task Auto_increment
title Varchar
FrK id_activity Int
FrK id_category Int
FrK id_user Int
FrK id_account Int
contain_task
PrK id_task Int
PrK id_journal_entry Int


JournalEntry
-Visibility
-DatePost
-Label
-Content
ItemJournalEntry
< Comment
< Create
< Comment
-Date
-Content
+notify()
Notification
-Content
-Target : Linkable
Receive
Profil
Account
-LastName
-FirstName
-Address
-Phone
-Email
-ID
-Password
-Login
Activity
-Title
-Description
Task
-Date[] = TimeSlot
-Title
-Description
Use v
Objective
-DeadLine : Date
-Title
-Description
Belong >
Belong >
0..1
0..1
Category
-Label
-Description
< Create
User
+CRUDCategory()
+CRUDCart()
+CRUDWishList()
+OrderCart()
+CRUDInput()
+RequestEstimate()
Seller
-SIRET
-WebSite
-Description
+DiscribeProfil()
+ProposeProductCategory()
+CRUDProduct()
+ProposeEstimate()
+AcceptCommand()
^ Manipulate (CRUD)
Administrator
+ManageAccount()
+CRUDProductCategory()
+CRUDActivityCategory()
Propose v
< (CRUD)
< (CRUD)
Tutorial
-Title
-Description
-Difficulty
Belong v
WishList
-Label
+AddProduct()
+RemoveProduct()
Cart
+Validate()
generate
Order
-Date
-TotalPrice
-DeliveryAddress
Estimate
-EstimatePrice
-Quantity
Propose v
< (CRUD)
BuyingProduct
-Quantity
-Attribute
-Price
refer
ActivityCategory
-Label
-ShortDescription
-Description
Belong >
Product
-Price
-StockQuantity
ProductCategory
-Label
-ShortDescription
-Description
Belong >
Content
-Quantity
-BuyPrice
< (CRUD)
Use >
< Create
Powered By Visual Paradigm Community Edition

Rapport personnel :

Rapport Personnel Loïc Dumas :

Lors de ce projet, j'ai eu l'occasion de jouer le rôle de manageur du projet, BD et des tests unitaires. Étant le manageur j'ai cherché à toujours motiver mes partenaires et organiser des réunions afin que chacun présente ses avancés et surtout présente ses problèmes, car je penses, que cela peut permettre d'avancer plus rapidement lorsque nous sommes bloqués (grâce à l'aide des autres qui peuvent voir rapidement une erreur que l'on a cherchée pendant longtemps.

Je considère avoir appris énormément durant ce projet. Je venais d'une filière Peip, si bien que j'ai eu très peu d'expérience en développement. À cause de cela, malgré les cours de programmation suivis à Polytech, je n'avais jamais eu l'occasion de travailler sur un projet de grande taille avec d'autres personnes. Avant ce projet, je ne considérais pas savoir développer une application.

Premièrement, mes plus grands acquis durant ce projet et au niveau programmation. Je considère mettre vraiment amélioré en java. Au début j'ai eu beaucoup de mal à commencer à coder, absolument pas par manque de motivation, mais ne sachant pas par où commencer. J'ai compris de nombreuses choses telles que les exceptions. Ensuite, j'ai pu approfondir mes connaissances avec Github. Lors de la phase de développement, j'ai principalement travaillé sur les wishList et le shop vu par l'utilisateur.

J'ai également pu mettre en pratique les design pattern que je n'avais jusqu'alors vu qu'en cours, et je n'avais pas aussi bien saisi leurs fonctionnements que maintenant, j'ai surtout pu avoir un exemple concret d'utilisation. Pour l'architecture de l'application, j'ai eu beaucoup de difficulté au début à comprendre "l'utilité" de toutes ces classes, et il m'aurait été impossible d'arriver à un tel résultat sans l'aide de nos encadrants et mes compagnons.

Durant ce projet, j'aurais apprécié arriver à développer une interface plus propre, ainsi que plus de fonctionnalité. Mais j'ai essayé de me concentrer sur : faire un programme le plus fonctionnel possible. Ensuite, surement en raison de mon inexpérience, je trouve être avancé trop lentement malgré tout les efforts impliqués, je reconnais un certaine frustration de ne pas être parvenu à faire tout ce que je souhaitais par cause de temps.

Concernant mon équipe, je trouve qu'il y a eu une très bonne cohésion, motivée et travailleur. On a jamais eu besoin de réclamer que quelqu'un travaille, étant donné que nous étions tous motivés pour progresser et apprendre !

La seule difficulté que j'ai rencontrée, et la difficulté de joindre tout le monde quelquefois, malgré que nous ayons planifié certains jours sur la fin du projet de faire des communications Skype pour partager notre avancement..

Conclusion, si la chose était à refaire, je ne pense pas qu'il y ait de chose majeure à changer. Et il s'agit pour moi sincèrement, du meilleur projet que j'ai eu à réaliser.

Rapport personnel : Pierre Casati

Lors de ce projet en groupe, j'ai été nommé responsable du design (architecture) de l'application, de la Javadoc, et de la soutenance.
Toutefois, de par l'expérience que j'avais déjà eu l'occasion d'acquérir, j'ai également servis de référent pour des problèmes liés à l'utilisation de Git, ou au concept de Programmation Orientée Objet.

Lors de la phase de design de l'application, il m'a fallu m'adapter. En effet, nous étions tous à niveau très inégal, et le but était que tout le monde comprenne et soit en accord avec les choix effectués. J'ai notamment pu m'apercevoir que même si dans ma tête certains concepts me paraissaient clairs, lorsque je devais les expliquer à une autre personne c'était complètement différent.

La partie Javadoc a sans doute été la partie la plus compliquée à gérer pour moi. J'ai en effet la mauvaise habitude de ne presque pas commenter mon code, on me l'a d'ailleurs fait remarquer !

Enfin, en ce qui concerne la préparation de la soutenance, il n'y avait pas de présentation orale proprement parlée à organiser, mais nous avons toutefois choisi de rediscuter des différents éléments du projet, à fin d'être sûr que tout le monde était au point et avait bien tout compris.

En ce qui concerne le projet de manière global, il m'a vraiment intéressé. Depuis le début du cursus ingénieur, c'était notre premier projet important de programmation en groupe et je l'attendais avec impatience. J'ai ainsi pu expérimenter de nouvelles architectures et découvrir de manière approfondie la gestion d'un groupe sur un projet dense.
Ce travail de groupe a d'ailleurs plutôt été un succès, puisqu'on a pu profiter des compétences de chacun. Le plus compliquer a finalement été de communiquer efficacement. En effet, beaucoup (et moi le premier), n'étaient pas disponibles de manière régulière. Cependant, nous étions tous motivés et intéressés, et cela a énormément compté dans l'avancée du projet.

En fin de compte, la seule partie où j'ai été déçu, c'est qu'au vu du nombre de fonctionnalités possibles, il a fallu établir des priorités. J'aurai souhaité pouvoir développer l'application en entier.

Petit point à noté : pendant le projet, j'ai écrit une petite « cheat sheet » pour git, que j'ai mis en accès libre pour le reste de la classe. Cela s'est avéré utile, puisque bon nombre d'élèves n'avaient pas eu tellement l'occasion d'utiliser cet outil de manière quotidienne avant ce projet.

Rapport personnel : Faustine Geoffray

J'ai beaucoup appris durant ce projet d'un point de vue technique. En effet, je n'avais jamais réellement pratiqué le langage Java, tout était donc nouveau pour moi. L'ambiance au sein de l'équipe était propice à l'apprentissage, ce qui m'a permis de comprendre un code et une architecture que je considère de complexe.

Durant ce projet, nous avons chacun développé un use case de notre côté, en essayant de ne pas modifier l'architecture de base. J'ai décidé de développer un use case facile, mais entièrement seule. Le use case que je devais développer est "updateAccount", use case dans lequel tout type de compte peut modifier ses informations personnelles. J'ai donc créé deux UI : une pour présenter les informations relatives au compte, et l'autre pour avoir la possibilité de modifier ces informations.

Les problèmes rencontrés durant la phase de développement ont été nombreux. Au début de la phase de développement, j'avais des difficultés à identifier le fonctionnement de l'architecture au sein d'Eclipse ; je me perdais très souvent à naviguer entre de nombreux fichiers. Je n'ai pas rencontré de problèmes particuliers de compréhension, mais de petites "broutilles" m'ont longuement freinées. J'ai perdu un temps fou à résoudre des erreurs que l'on pourrait qualifier de "langages". Par exemple, je n'arrivais pas à ajouter d'action performed sur mes boutons, ceci était dû à une mauvaise déclaration de ceux-ci. De même, je n'arrivais pas à récupérer l'identifiant du compte, ceci était dû à une erreur d'orthographe dans une variable... Au fur et à mesure du projet, j'ai appris à trouver les sources des erreurs, il m'est donc aujourd'hui plus simple de les résoudre.

J'ai aussi participé au développement de la base de données sur PostgreeSQL, dans laquelle j'ai créé et remplis quelques tables.

Concernant l'organisation du projet nous avons utilisé deux principales technologies de travail collaboratif, qui sont Github et Google Drive. Tout d'abord, nous avons utilisé Google Drive pour partager l'intégralité de nos documents formels (divers diagrammes et comptes rendus). Cette application nous a permis de réaliser la phase de modélisation et de conception sans trop de problèmes. Nous avons ensuite utilisé Github pour partager notre code durant la phase de développement. Nous avons couplé à Github l'extension WinMerge, qui nous a simplifié la gestion des conflits entre les documents lors des "git merge".
Nous organisions des réunions de travails hebdomadaires durant lesquelles nous faisions un point sur le travail fait et à faire, nous proposions des modifications pour améliorer le code et nous vérifions la bonne compréhension de l'ensemble du groupe sur notre travail. Ces réunions étaient très instructives et nous ont permis de mener au mieux notre projet.

Bien que je regrette le cours délai de la phase de développement - car la phase de prise en main d'Eclipse m'a beaucoup retardé - j'ai vraiment apprécié travailler sur ce projet. Notre équipe était complémentaire et j'ai appris beaucoup de choses, comme par exemple développer en java, faire des tests unitaires, créer de belles UI avec swing, trouver les sources de mes erreurs...

Rapport personnel : Nassim Vachor

Abstract

This paper lays out about my work and my contribution in this software engineering project. this project consists to design an application in java which help people to create their projetcs by themselves.
In fact, after a constitution of subgroups, I have worked specialy on Seller Use case.
So, i have developed the Describe Profil use case of seller and the manage shop use case.
Firstly, i began with the design of class diagramme of each use case and after i have started the developement.
Personally, I think this project was useful. It permitted me to have a better understanding of the the MVC and 3tiers architectur and i have learned a lot of concept in java such as the factory design pattern, how to use it, how to link methods with the database, connect to the database...
i have also learned how to work with team and manage a group of individuals.

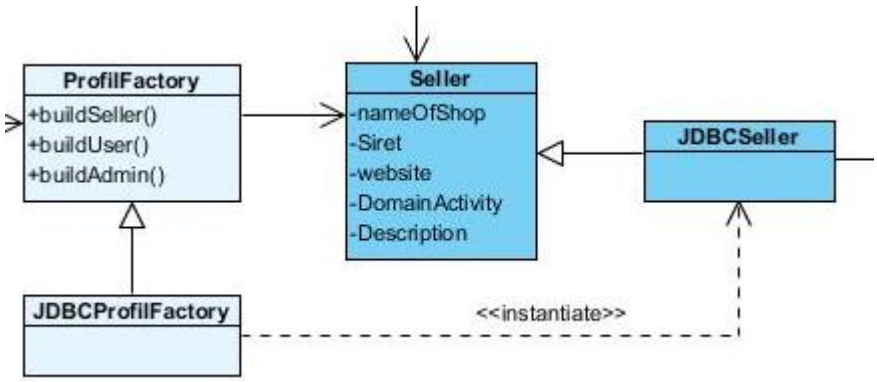
Introduction

Durant ce projet j'ai des différents rôle au sein de notre équipe, ce qui m'a permis a touché à plusieurs domaines et à me perfectionner particulièrement en développement et en tous ce qui est technique.
Dans un premier, j'étais responsable de Modélisation et de Use cases, un domaine que je maîtrise assez bien et cela à travers les différents cours de modélisation que j'ai eu durant mon cursus universitaire.
Pour mieux s'organiser, nous avons définit un tableau contenant les différents uses cases à modéliser. Ce tableau m'a servi particulièrement à répartir les différents uses et à en attribuer à chaque membre.
Connaissant que l'architecture MVC présenté au cours de java au premier semestre, j'ai eu quelques difficultés dans un premier temps à concevoir mon diagramme de classe et à trouver la bonne architecture. Ainsi, j'ai essayé de faire un modèle selon l'architecture MVC, cependant après la réunion du groupe , nous avons constaté qu'il fallait mieux partir sur l'architecture 3-tiers, qui se partage sur trois niveau: persistance (accès à la BD et aux données), logique (qui s'occupe de tous ce qu est calcul) et la couche présentation qui s'agit des différents vues (UI).

Travail réalisé

Une fois que la modélisation est terminée, je me suis penché sur la couche persistance et à essayer de comprendre comment la base interagit avec les méthodes et comment la connexion est établie.
Avec mon équipe, on a réussi à établir la connexion avec une base de données locale dans un premier temps, ensuite on l'a mis sur un serveur distant "Heroku", ce qui nous a permis

de faire toutes les créations des tables et toutes les insertions nécessaires et avoir tous le même état de notre base.
Ensuite, j'ai contribué aux développements du Use Case Login, qui nous a servi comme modèle pour la suite du développement de notre application.
Après cela je me suis penché sur le développement de mes uses cases, qui sont comme cité auparavant "DescribeProflSeller" et "ManageShop".
le premier permet au seller, de modifier ces informations personnelles et de les mettre à jour. Donc je me suis occupé dans un premier de la création de la table Seller dans la BD, ensuite j'ai défini la classe abstract Seller qui contient les différent attribut et getter et setter et qui comme classe fille JDBCSeller (établis le lien avec la BD)
ensuite, j'ai procédé à la mise en place du design Abstract Factory, un design que je ne maitrisais pas trop avant ce projet, ainsi j'ai pu créer les différentes Factory nécessaires à la création de mon seller.



Ensuite, j'ai développé le UpdateProfilHandler, qui manipule un objet de type seller et qui contient toutes les méthodes nécessaires. Ensuite je suis passé à la Facade qui sépare l'interface graphique et la conception. Ainsi la classe Facade délègue et fait appel aux méthodes présentes dans le Handler.
Ensuite j'ai procédé au développent des mes UI, et de faire lien avec les méthodes nécessaires lors de l'édition du Seller Profil.
En ce qui concerne le deuxième use case "manage Shop", qui permet au seller de gérer sa boutique c'est d'afficher tous ces produits avec toutes les informations relatives à ce produit ainsi que de pouvoir rajouter des nouveaux produits et d'en supprimer ou modifier d'autres. j'ai procédé au développement de ce use case de la même façon que le premier sauf que dans ce cas j'ai eu besoin d'un Set qui est un ensemble de produits d'un seller.
Enfin, à la fin de ce projet j'ai pu constater une grande marge de progression au niveau de mes compétences en développement java et en programmation-objet en général, j'aurais bien aimé avoir plus de temps afin de perfectionner et d'organiser mieux mon code, car au début j'avançais lentement, mais ces derniers jours je commence à maitriser le sujet ainsi je suis plus efficace.