

UNIVERSITÉ NATIONALE DU VIETNAM À HANOÏ
INSTITUT FRANCOPHONE INTERNATIONAL



ĐẠI HỌC QUỐC GIA HÀ NỘI
VNU
Since 1906



Option : Systèmes Intelligents et Multimédia (SIM)

Promotion : XXI

Génie logiciel avancé
Rapport du Projet en Binome

**Étude et mise en place d'un système de payement
électronique dans la Résidence universitaire Ky Tuc Xa
My Dinh 2**

DIALLO Azise Oumar

RAKOTOARIVELO Nobby

Encadrant :

Dr HO Tuong Vinh, Professeur (VNU, IFI)

Année académique 2016-2017

Table des matières

| | | |
|----------|--|-----------|
| 1 | Introduction générale | 2 |
| 2 | Présentation du projet : la problématique | 2 |
| 2.1 | Contexte | 2 |
| 2.2 | Problématique | 2 |
| 2.3 | Objectif. | 3 |
| 3 | Les exigences fonctionnelles et non fonctionnelles | 3 |
| 3.1 | Les exigences fonctionnelles | 3 |
| 3.2 | Quelques scénarios des fonctionnalités | 6 |
| 3.3 | Les exigences non fonctionnelles | 7 |
| 4 | Conception du Système | 7 |
| 4.1 | Architecture du système | 7 |
| 4.2 | Diagramme de classes. | 8 |
| 4.3 | Quelques diagrammes de séquences. | 10 |
| 5 | Implementation | 10 |
| 5.1 | Environnement matériel | 10 |
| 5.2 | Environnement logiciel | 11 |
| 5.2.1 | Langages de programmation | 11 |
| 5.2.2 | Les plateformes utilisées | 11 |
| 5.2.3 | Tests et mesures du logiciel | 11 |
| 6 | Conclusion générale | 14 |
| 7 | Annexe : Présentation de l'application | 15 |
| 7.1 | Premier lancement de l'application | 15 |
| 7.2 | Accueil Administration | 16 |
| 7.3 | Accueil Gestionnaire | 16 |
| 7.4 | Accueil Étudiant | 16 |
| 7.5 | Opérations sur les comptes client | 16 |
| 7.6 | Effectuer un versement (dépôt) | 16 |
| 7.7 | Confirmation d'un versement (dépôt) | 19 |
| 7.8 | Consulter solde | 19 |
| 7.9 | Effectuer un paiement | 19 |
| 7.10 | Confirmation de paiement | 21 |
| 7.11 | Historiques des paiements reçus par un prestataire | 21 |

1 Introduction générale

Dans le cadre du cours « module Génie logiciel avancé », nous avons reçu un projet de développement logiciel. Ce projet fait suite au premier travail pratique (TP1) dont le but était de nous rappeler les concepts de la modélisation avec UML et la programmation orientée-objet avec Java.

Ce projet a pour objectif principal de mettre en oeuvre tous les processus dans le cadre de développement d'un logiciel. A cet effet, il a été demandé à chaque groupe de choisir un projet à réaliser. Ainsi, notre choix s'est porté sur l'étude et la mise place d'un système de paiement électronique autour du thème : « l'étude et la mise en place d'un système de paiement électronique dans la Résidence universitaire de My Dinh2. » Ce choix du projet n'est pas fortuit. En effet, le phénomène du paiement électronique prend de plus en plus d'envergure et présente de nombreux défis à réaliser. Par ailleurs, notre projet doit répondre à un besoin réel de la Résidence.

La suite de ce présent rapport consistera à présenter les différentes étapes du développement de notre logiciel à savoir l'analyse, la conception et le développement. Nous allons conclure après avec une auto-évaluation de notre travail.

2 Présentation du projet : la problématique

2.1 Contexte

La résidence KTX My Dinh fait partie du Centre de Service des étudiants de l'Université nationale du Vietnam à Hanoï.

Missions. La résidence a pour missions d'organiser la réception, les arrangements d'hébergement pour les étudiants et les étrangers au Vietnam conformément aux normes, aux réglementations des autorités et aux autorités compétentes VNU ;

Prestations. La résidence fournit la gestion et l'exploitation de condominiums, de bâtiments et des opérations commerciales, des services pour la vie quotidienne, académique, scientifique et de divertissement pour les étudiants.

2.2 Problématique

La résidence KTX My Dinh offre plusieurs types de prestations (restauration, parking, boutique...) qu'elle soustrait avec des Partenaires (prestataires). Par ailleurs, elle enregistre un effectif de plus 50 000 étudiants. On peut ainsi facilement percevoir les difficultés de gestion des services fournis aux étudiants par l'intermédiaire des Prestataires. En effet, il est presque impossible d'avoir une idée des paiements effectués dans cet écosystème. Le gestionnaire n'a aucune vue globale des transactions effectuées. Par ailleurs, les étudiants sont obligés de souscrire séparément aux différents abonnements proposés par chaque prestataire. Ainsi, un étudiant peut se retrouver avec plus de cinq cartes pour jouir des services de la Résidence. D'où la problématique

« Comment payer tous les services proposés dans la Résidence par l'intermédiaire d'une unique carte notamment la carte de Résidence ? »

2.3 Objectif.

L'objectif général vise par ce projet est tout d'abord d'étudier et développer un système afin de faciliter et d'améliorer les opérations de paiement entre le Service Comptabilité de la Résidence, les Prestataires et les Étudiants. Spécifiquement, ce projet vise à analyser l'existant et concevoir un système complet et sécurisé pour le paiement électronique au sein de la Résidence.

Cependant, il faut signaler qu'au regard de la taille du projet et du temps de travail limité, nous nous sommes donné pour objectif de concevoir le système en adoptant une approche évolutive (modèle évolutif). En effet, il s'agit ici d'un développement exploratoire afin d'avoir une idée précise d'un tel système.

3 Les exigences fonctionnelles et non fonctionnelles

Dans cette partie, nous nous intéressons à l'analyse des besoins c'est à dire la description et la spécification du système logiciel à développer. Il existe deux types d'exigences à savoir fonctionnelles et non fonctionnelles.

Après avoir identifié les besoins du Gestionnaire de la Résidence de KTX MD2, nous avons procédé à la définition des composantes logicielles qui réalisent les fonctionnalités souhaitées par les utilisateurs. Ainsi, ces composants qui seront définis, permettront de réaliser le système avec suffisamment de précision. Pour cela, nous utiliserons les diagrammes UML pour avoir une vue détaillée du système. Pour ce faire, nous utiliserons principalement le diagramme de cas d'utilisation.

3.1 Les exigences fonctionnelles

Il est important d'avoir à l'esprit qu'un système de paiement doit être un système qui supporte la gestion automatique des clients et de leurs comptes.

Nous allons utiliser le diagramme de cas d'utilisation (Use Case) pour illustrer les exigences fonctionnelles de notre application. Ce diagramme est un moyen d'exprimer les besoins des utilisateurs d'un système informatique. Dans notre système actuel, nous avons identifié trois (03) utilisateurs principaux à savoir le Gestionnaire, le prestataire et l'étudiant. Le tableau (TABLE 1) résume les exigences fonctionnelles de notre application. Il représente le « Product backlog » en utilisant notamment la méthode SCRUM.

Ainsi, de l'analyse du tableau (TABLE 1), nous avons construit le diagramme de cas d'utilisation de notre système présenté dans la figure (FIGURE 1).

Droits d'accès. Il y a trois principaux niveaux définis en fonction du statut de l'utilisateur définis comme suit :

TABLE 1 – Les exigences fonctionnelles de l’application (Product backlog)

| ID | Nom | Priorité | Coût | Comment démontrer | Observations |
|----|-------------------------|----------|------|---|---------------------------------|
| 1 | Créer un compte | 50 | 2 | Connectez-vous, ouvrez la fenêtre de création de compte, créer le compte, afficher la liste des comptes créés et vérifiez que le compte est bien présent | Besoin du diagramme de séquence |
| 2 | Supprimer un Compte | 30 | 1 | Connectez-vous, afficher la liste des comptes créés, supprimer le compte et vérifiez que le compte est bien supprimé dans la liste | Besoin du diagramme de séquence |
| 3 | Modifier compte | 30 | 3 | Connectez-vous, afficher la liste des comptes créés, modifier le compte sélectionné et vérifiez que le compte est bien modifié dans la liste | Besoin du diagramme de séquence |
| 4 | Consulter solde | 50 | 3 | Connectez-vous, ouvrez la fenêtre de consultation, préciser le numéro de compte | Besoin du diagramme de séquence |
| 4 | Effectuer paiement | 50 | 5 | Connectez-vous, ouvrez la fenêtre de consultation, préciser le numéro de compte | Besoin du diagramme de séquence |
| 5 | Historiques | 50 | 5 | Connectez-vous, ouvrez la fenêtre d’historiques, préciser le numéro de compte et le mot clé de la recherche | Besoin du diagramme de séquence |
| 6 | Créditer compte | 40 | 2 | Connectez-vous, ouvrez la fenêtre de recherche de compte, effectuer le versement et vérifiez que le compte est bien crédité du montant versé | Besoin du diagramme de séquence |
| 7 | Opérations gestionnaire | 40 | 3 | Connectez-vous, afficher la liste des opérations, rechercher les opérations en fonction du mot clé | Besoin du diagramme de séquence |
| 8 | Créer un utilisateur | 50 | 2 | Connectez-vous, ouvrez la fenêtre de création d’utilisateur, créer l’utilisateur, afficher la liste des utilisateurs créés et vérifiez que l’utilisateur est bien présent | Besoin du diagramme de séquence |
| 9 | Supprimer utilisateur | 30 | 1 | Connectez-vous, afficher la liste des utilisateurs créés, supprimer l’utilisateur et vérifiez que l’utilisateur est bien supprimé dans la liste | Besoin du diagramme de séquence |
| 10 | Modifier utilisateur | 30 | 3 | Connectez-vous, afficher la liste des utilisateurs créés, modifier l’utilisateur sélectionné et vérifiez que l’utilisateur est bien modifié dans la liste | Besoin du diagramme de séquence |

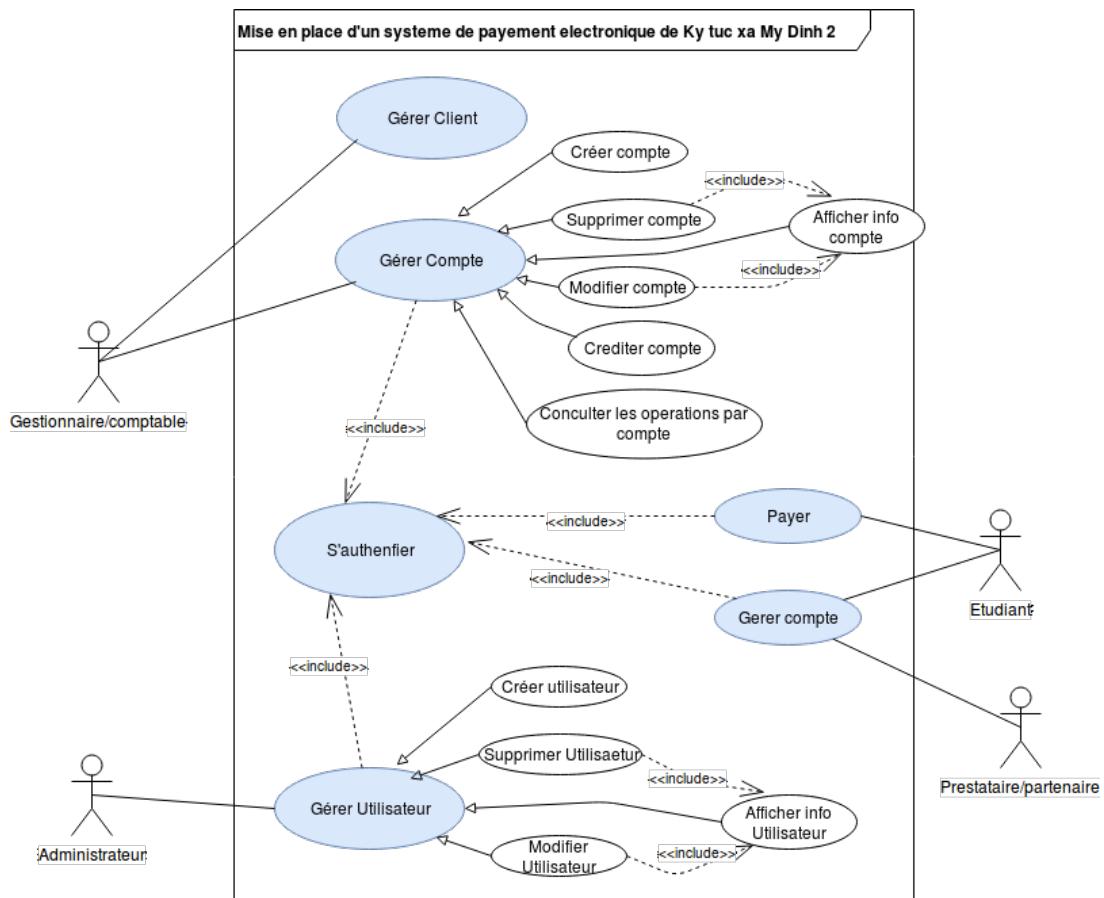


FIGURE 1 – Diagramme de cas d'utilisation

TABLE 2 – Scénario d’authentification

| | |
|---|---|
| 1. L’utilisateur renseigne le login et le mot de passe | 2. Le système vérifie l’identité de l’utilisateur |
| 3. Si les informations sont incorrectes, alors retour au point1 | 4. Sinon, le système affiche la page d’accueil correspondant au profil. |

TABLE 3 – Scénario de création d’un compte

| | |
|--|---|
| 1. Le gestionnaire renseigne son login et son mot de passe | 2. Le système vérifie l’identité de l’utilisateur |
| 3. Le système affiche la page d’accueil | 4. L’utilisateur demande la page de création de compte. |
| 5. Le système affiche la page de création de compte | 6. L’employé renseigne les informations sur le compte ainsi que sur le propriétaire du compte s’il est nouveau sinon trouver d’abord le propriétaire qui veut créer un compte |
| 7. Le système vérifie les données saisies et renvoie la confirmation de création | |

- Niveau 1 : Statut Administrateur : c’est l’utilisateur principal en charge de la création des autres profils d’utilisateurs.
- Niveau 2 : Statut Gestionnaire (Employé de la Résidence) charge de la gestion des clients et de leurs comptes.
- Niveau 3 : Statut Client. Il y a deux types de clients notamment les étudiants et les prestataires. Chaque type a des droits différents.

Chaque niveau donne accès aux différentes actions suivant les niveaux dans le système.

Ainsi, un profil administrateur pourra donc créer, modifier ou supprimer d’autres utilisateurs. Il pourra également gérer le système et modifier les paramètres de celui ci. Ensuite, un employé peut par exemple gérer les clients et effectuer des opérations sur leurs comptes. Cependant, il n’aura pas accès à la gestion des utilisateurs du système. Quant au client, il pourra consulter son solde, effectuer des paiements ainsi que consulter l’historique des paiements.

Par ailleurs, pour des mesures de sécurité, l’accès à l’application est interdit pour tous autres utilisateurs hormis les employés, le client et l’administrateur.

Dans la partie 4, nous allons présenter l’architecture du système.

3.2 Quelques scénarios des fonctionnalités

Scénario de cas d’utilisation « authentification ». Voir le tableau (TABLE 2)

Scénario de cas d’utilisation « Création de compte ». Voir le tableau (TABLE 3)

TABLE 4 – *Scénario de paiement*

| | |
|---|---|
| 1. Le client renseigne son login et son mot de passe | 2. Le système vérifie l'identité du client |
| 3. Le système affiche la page d'accueil | 4. L'utilisateur demande la page de paiement. |
| 5. Le système affiche la page de paiement | 6. Le client renseigne les informations sur le paiement |
| 7. Le système vérifie les données saisies notamment le solde du compte du client et renvoie la confirmation de paiement ou une message d'erreur le cas échéant. | |

TABLE 5 – *Les exigences non fonctionnelles de l'application*

| Exigences | Description | Exigences fonctionnelles requises |
|---------------|--|-------------------------------------|
| Disponibilité | Le système doit être disponible durant toutes les opérations | Toutes les exigences fonctionnelles |
| Disponibilité | Les données doivent être disponibles et accessibles à tout moment où l'utilisateur en a besoin | Toutes les exigences fonctionnelles |
| Intégrité | En cas de défaillance du système, les données ne doivent pas subir un changement | Toutes les exigences fonctionnelles |
| Performance | Le Système doit répondre rapidement aux demandes de l'utilisateur | Toutes les exigences fonctionnelles |
| Sécurité | Le Système doit être sécurisé à tous les points car c'est de la gestion de l'argent qu'il s'agit | Toutes les exigences fonctionnelles |

Scénario de cas d'utilisation « Effectuer paiement ». Voir le tableau (TABLE 4)

3.3 Les exigences non fonctionnelles

Ce sont des exigences liées notamment à la performance, la sûreté, la confidentialité, la sécurité... du système. De plus, il faut chercher des critères mesurables de l'application. Le tableau (TABLE 5) regroupe les exigences non fonctionnelles de notre application.

4 Conception du Système

4.1 Architecture du système

Suite à la phase d'analyse des exigences fonctionnelles de l'application, nous avons opté pour une application Web. Pour ce faire, nous allons utiliser une « architecture MVC (Modèle-Vue-Contrôleur) » pour notre application (voir FIGURE 2).

TABLE 6 – Description des couches de l’architecture du système

| Num | Couches | Description | Composants | Outils et technologies |
|-----|------------|--|--|------------------------|
| 1 | VUE | La Vue représente la couche avec laquelle l’utilisateur interagit | Feuilles de styles, pages HTML | HTML, BootStrap |
| 2 | CONTROLEUR | Le Contrôleur peut communiquer avec la Vue et le Modèle afin de traiter les requêtes des clients et d'accéder à la base de données | serveur d’application (conteneur Web), DAO (pour l'accès aux données), les entités (classes), les interfaces, les implémentations, les vues JSP, les connexions et les tests | Tomcat 8.5, JDBC, JSP |
| 3 | MODELE | Cette représente les données que le système manipule et qui sont stockées dans une Base de données | BD relationnelle | MySQL |

Puisqu'il s'agit d'une application Web, elle doit disposer d'un serveur Web auquel se connecteront des utilisateurs à travers des clients légers tels les navigateurs Web qui sont disponibles quasiment sur tous les systèmes d'exploitation modernes. Ainsi, un utilisateur peut accéder à l'application indépendamment des contraintes environnementales. Les différents composants de l'architecture sont présentés dans la FIGURE 2.

L'architecture MVC est à 3-tiers ou à trois couches [4]. Chaque couche est définie dans le tableau ci-après (6). Pour la définition des couches, nous nous sommes inspirés des travaux de la société IMPROVE [2].

Chaque couche ne communique qu'avec les couches adjacentes. Ainsi, les données sont préservées des modifications des utilisateurs d'une part. De plus, cette modularisation du système en couches permet de développer chacune de ces couches indépendamment des autres.

4.2 Diagramme de classes.

Après avoir illustré les besoins des utilisateurs de notre système par le diagramme de cas d'utilisation, nous passons à l'aspect conceptuel avec un des diagrammes structurels à savoir le diagramme de classes. Il est généralement considéré comme le plus important dans le développement orienté objet. Il représente l'architecture conceptuelle de notre système (Voir FIGURE 3).

Pour notre système, nous avons identifié principalement douze (06) classes dont nous faisons quelques descriptions ci-dessous :

- La classe Client. Une instance peut être soit un étudiant soit un prestataire. Un prestataire peut avoir un ou plusieurs comptes.
- La classe Compte. Un compte est détenu par un et seul Client.

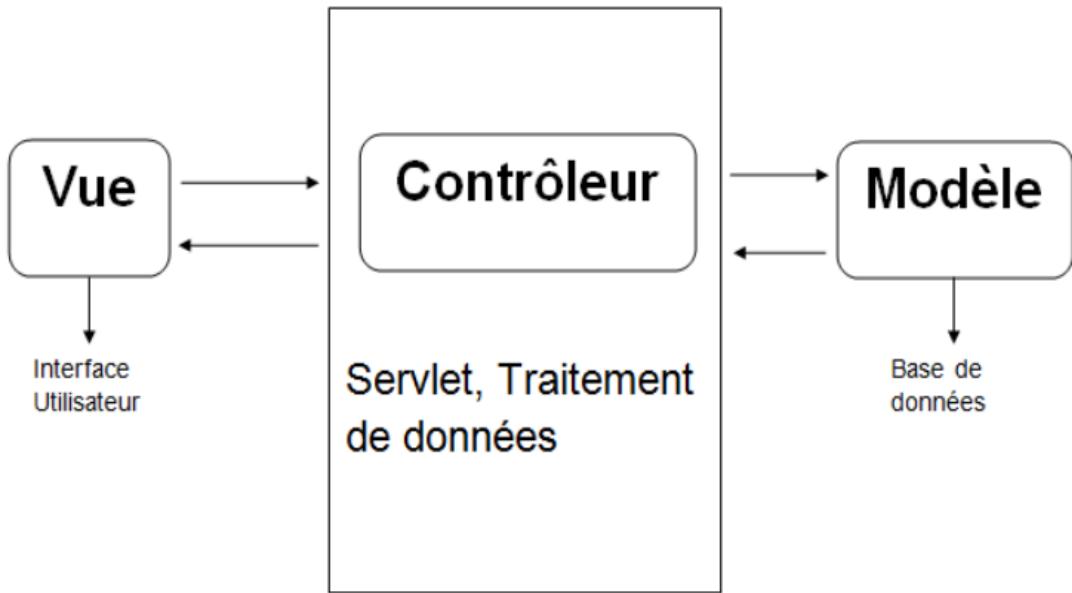


FIGURE 2 – Architecture MVC

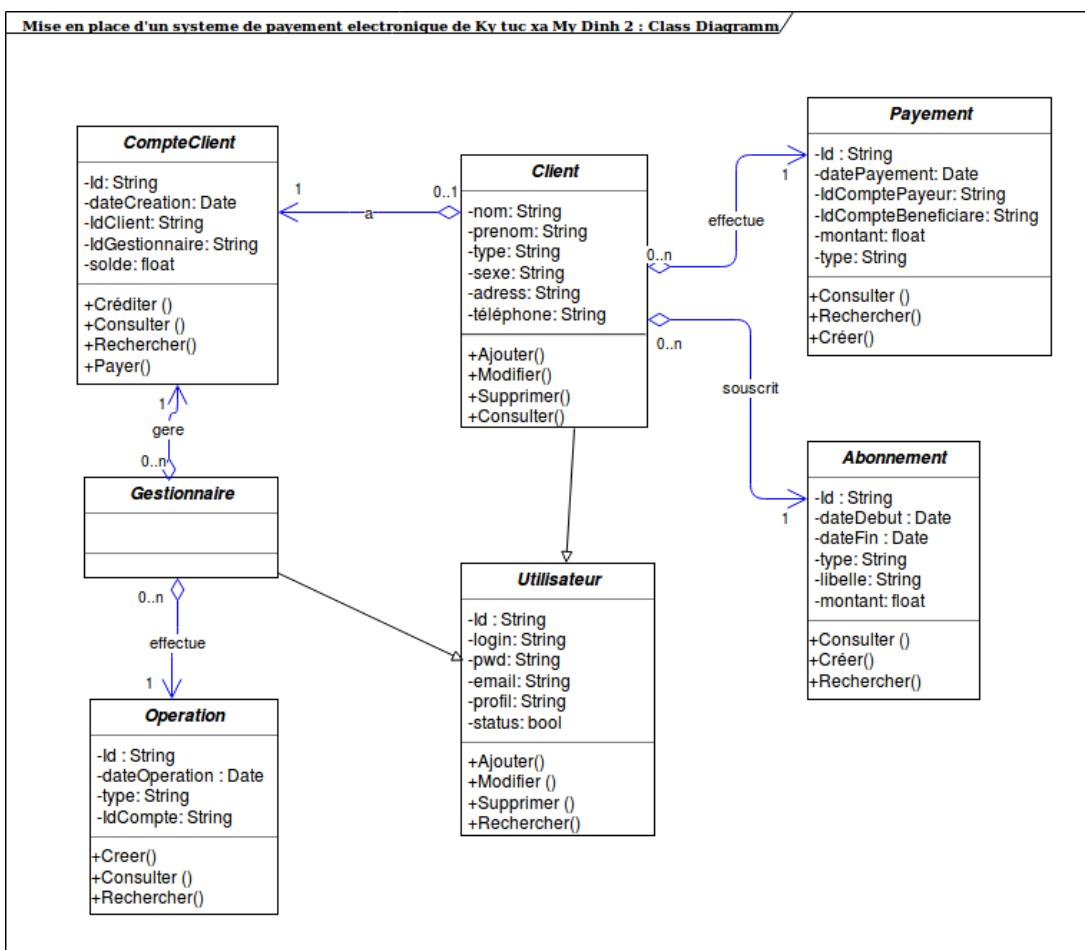


FIGURE 3 – Diagramme de classe

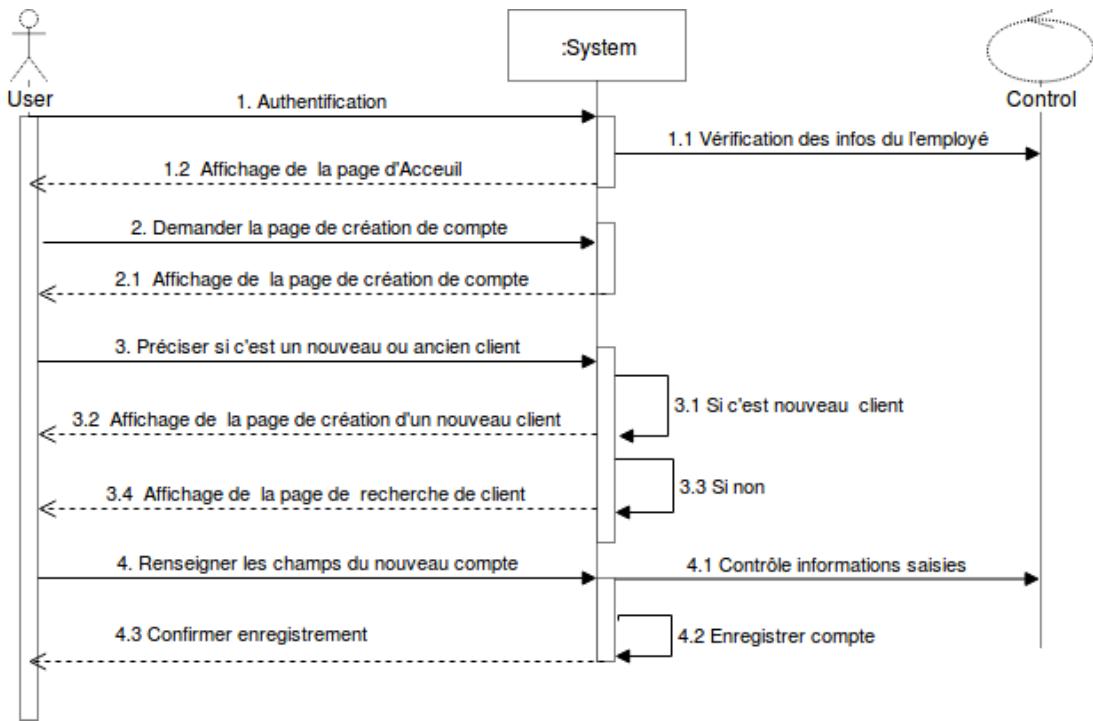


FIGURE 4 – Diagramme de séquence : Créeer compte client

- La classe Utilisateur. C'est cette classe qui va définir les profils des utilisateurs du système. Plusieurs utilisateurs peuvent avoir plusieurs profils mais un seul profil est attribué à un utilisateur.

4.3 Quelques diagrammes de séquences.

Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur, ce qui fait de lui un des diagrammes d'interaction (dynamique).

Les FIGURES 4 5 illustrent quelques interactions avec le système.

5 Implementation

5.1 Environnement matériel

Pour le développement et les expérimentations nous allons utiliser un ordinateur portable avec les caractéristiques suivantes :

- Processeur : Intel(R) CoreT M i5-M370 @2.4 GHZ
- RAM : 8.00 Go
- OS : Ubuntu 16.04
- JDK8

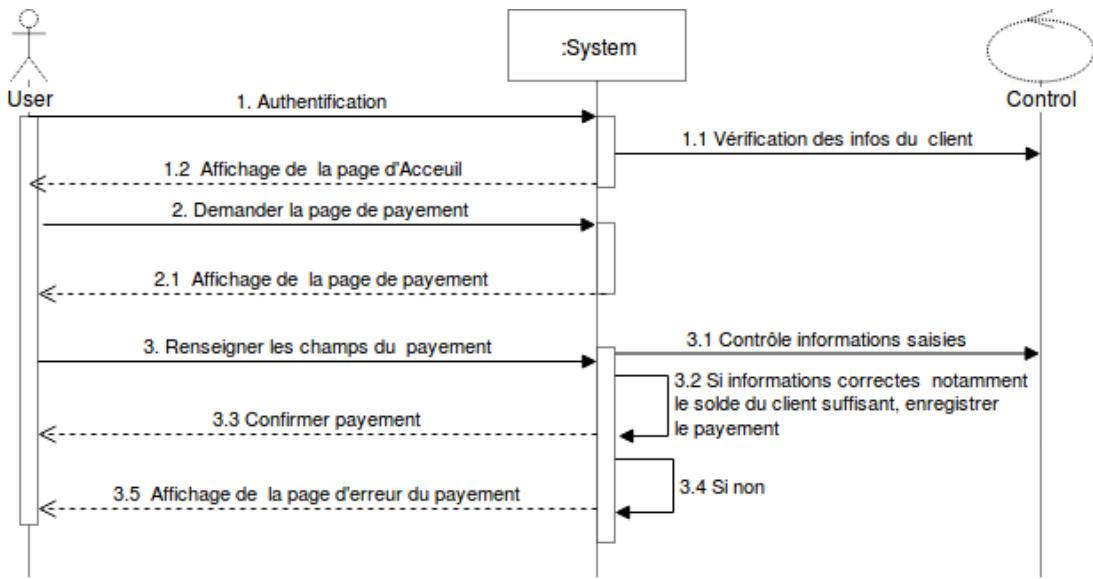


FIGURE 5 – Diagramme de séquence : Effectuer paiement

5.2 Environnement logiciel

5.2.1 Langages de programmation

Le système de paiement sera codé en Java Web. Il s'agit d'une application Web dynamique. Pour ce faire, nous utilisons l'environnement de développement intégré (IDE) Eclipse (Neon3). Le choix du langage Java n'est pas fortuit. En effet, en plus d'être un choix de l'équipe projet, ce langage est le mieux pour illustrer la programmation orientée objet (POO).

Nous tenons à signaler ici que nous n'avons pas utilisé de FrameWork spécifique. Tout le développement a été fait manuellement que ce soit les contrôleurs et les accès la base de données.

5.2.2 Les plateformes utilisées

Plusieurs plateformes ont été utilisées durant le présent projet à savoir :

- Github (<https://github.com/diallitoz/ProjetGLA2017>) : Pour l'hébergement et la gestion de développement de l'application [3].
- draw.io : Pour la conception des diagrammes UML [1].
- ShareLatex (<https://fr.sharelatex.com/>) : Pour la rédaction du rapport [5].

5.2.3 Tests et mesures du logiciel

Tests unitaires. Pour les tests unitaires, nous utilisons l'outil *Junit*.

Nous avons implémenté plusieurs tests unitaires dont nous présentons trois (03) cas dans ce rapport. Il s'agit de la création d'un compte, du paiement et du test général (voir (FIGURES 6, 7, 8)).

The screenshot shows the Eclipse IDE interface with the JUnit perspective selected. The top part displays the code for `CreationCompteTest`:

```

1 package dao;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import metier.entities.Compte;
6
7 public class CreationCompteTest {
8
9     @Test
10    public void testCreationCompte() {
11        CompteDaoImplementation dao = new CompteDaoImplementation();
12        Compte cptel = dao.enregistrerCompte(new Compte("Prestataire", "2017.11.07", 3, 2, 50000.0));
13        assertEquals("Prestataire", cptel.getType());
14    }
15 }

```

The bottom part shows the JUnit results:

- Runs: 1/1
- Errors: 0
- Failures: 0

The status bar indicates "Finished after 0,385 seconds".

FIGURE 6 – Test unitaire : création d'un compte

The screenshot shows the Eclipse IDE interface with the JUnit perspective selected. The top part displays the code for `PayementTest`:

```

1 package dao;
2
3 import static org.junit.Assert.*;
4 import java.util.Date;
5 import org.junit.Test;
6 import metier.entities.Compte;
7 import metier.entities.Payement;
8
9 public class PayementTest {
10
11     @Test
12    public void testPayement() {
13        Payement payement = new Payement();
14        CompteDaoImplementation dao = new CompteDaoImplementation();
15        Compte cptel = dao.enregistrerCompte(new Compte("Prestataire", "2017.11.07", 3, 2, 50000.0));
16        Compte cptel2 = dao.enregistrerCompte(new Compte("Etudiant", "2017.10.27", 30, 3, 150000.0));
17
18        payement = dao.EffectuerPayement(cptel.getId(), cptel.getType(), 45000.0, new Date().toString(), "Pressing");
19
20        assertEquals("Pressing", payement.getType());
21    }
22 }

```

The bottom part shows the JUnit results:

- Runs: 1/1
- Errors: 0
- Failures: 0

The status bar indicates "Finished after 0,608 seconds".

FIGURE 7 – Test unitaire : effectuer payement

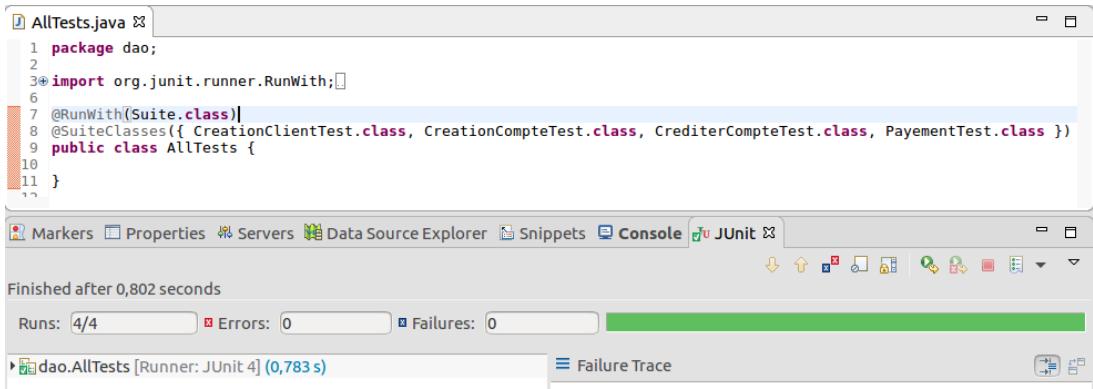


FIGURE 8 – Tous les tests unitaires

```

1 src/dao/AllTests.java:9:  Each class should declare at least one constructor
2 src/dao/AllTests.java:9:  headerCommentRequirement Required
3 src/dao/ClientDaoImplementation.java:12:  Each class should declare at least one constructor
4 src/dao/ClientDaoImplementation.java:12:  headerCommentRequirement Required
5 src/dao/ClientDaoImplementation.java:15:  Parameter 'client' is not assigned and could be declared final
6 src/dao/ClientDaoImplementation.java:15:  publicMethodCommentRequirement Required
7 src/dao/ClientDaoImplementation.java:16:  Ensure that resources like this Connection object are closed after use
8 src/dao/ClientDaoImplementation.java:16:  Local variable 'connexion' could be declared final
9 src/dao/ClientDaoImplementation.java:18:  Avoid variables with short names like ps|
10 src/dao/ClientDaoImplementation.java:18:  Local variable 'ps' could be declared final
11 src/dao/ClientDaoImplementation.java:18:  Potential violation of Law of Demeter (object not created locally)
12 src/dao/ClientDaoImplementation.java:22:  Potential violation of Law of Demeter (object not created locally)
13 src/dao/ClientDaoImplementation.java:23:  Potential violation of Law of Demeter (object not created locally)
14 src/dao/ClientDaoImplementation.java:24:  Potential violation of Law of Demeter (object not created locally)
15 src/dao/ClientDaoImplementation.java:25:  Potential violation of Law of Demeter (object not created locally)
16 src/dao/ClientDaoImplementation.java:26:  Potential violation of Law of Demeter (object not created locally)
17 src/dao/ClientDaoImplementation.java:27:  Potential violation of Law of Demeter (object not created locally)
18 src/dao/ClientDaoImplementation.java:28:  Potential violation of Law of Demeter (object not created locally)
19 src/dao/ClientDaoImplementation.java:29:  Potential violation of Law of Demeter (object not created locally)
20 src/dao/ClientDaoImplementation.java:30:  Potential violation of Law of Demeter (object not created locally)
21 src/dao/ClientDaoImplementation.java:31:  Potential violation of Law of Demeter (object not created locally)
22 src/dao/ClientDaoImplementation.java:32:  Potential violation of Law of Demeter (object not created locally)
23 src/dao/ClientDaoImplementation.java:33:  Potential violation of Law of Demeter (object not created locally)
24 src/dao/ClientDaoImplementation.java:35:  Potential violation of Law of Demeter (object not created locally)
25 src/dao/ClientDaoImplementation.java:36:  Local variable 'ps2' could be declared final
26 src/dao/ClientDaoImplementation.java:36:  Potential violation of Law of Demeter (object not created locally)
27 src/dao/ClientDaoImplementation.java:37:  Avoid variables with short names like rs|
28 src/dao/ClientDaoImplementation.java:37:  Ensure that resources like this ResultSet object are closed after use
29 src/dao/ClientDaoImplementation.java:37:  Local variable 'rs' could be declared final
src/dao/ClientDaoImplementation.java:37:  Potential violation of Law of Demeter (object not created locally)

```

FIGURE 9 – Extrait du rapport de détection de bugs PMD

Tests de détection de bugs. Pour les tests de détection de bugs, nous utilisons l’outil *PMD* (*Programming Mistake Detector*).

Pour ce faire, nous avons lancé plusieurs fois cet outil afin de corriger le programme (voir (FIGURE 9) pour un extrait du rapport). Le rapport global se trouve dans le dossier du projet. Ainsi, nous avons corrigé par exemple le nom des variables et l’endroit de leur déclaration.

Mesure du logiciel. Nous avons utilisée *Eclipse metrics* pour la mesure de qualité de notre logiciel.

La (FIGURE 10) nous donne un extrait du rapport. Le rapport global se trouve dans le dossier du projet. Grâce à cet outil, nous avons un peu amélioré la qualité de notre application notamment sur la répétitions de codes et l’utilisation des boucles. Cependant, des améliorations restent toujours à faire surtout au niveau du contrôleur (Servelet). En effet, n’utilisant pas de framework, nous avons du gérer cas par cas les requêtes au serveur. Ce qui a été coûteux en terme « cyclomatique ».

| Metric | Total | Mean | Std. Dev. | Maximum | Resource causing Maximum | Method |
|--|-------------|--------|-----------|---------|--|--------|
| Number of Parameters (avg/max per method) | 1 | 0,794 | 1,31 | 12 | /ProjetGLA2017/src/metier/entities/Client.java | |
| Number of Static Attributes (avg/max per class) | 1 | 0,043 | 0,204 | 1 | /ProjetGLA2017/src/dao/SingletonConnexion.java | |
| Efferent Coupling (avg/max per package) | 6,667 | 5,735 | | 14 | /ProjetGLA2017/src/dao | |
| Specialization Index (avg/max per type) | 0,141 | 0,61 | | 3 | /ProjetGLA2017/src/web/ControleurServlet.java | |
| Number of Classes (avg/max per package) | 23 | 7,667 | 2,357 | 11 | /ProjetGLA2017/src/dao | |
| Number of Attributes (avg/max per type) | 63 | 2,739 | 3,3 | 13 | /ProjetGLA2017/src/metier/entities/Client.java | |
| Abstractness (avg/max per package/file) | 0,089 | 0,126 | 0,267 | 1 | /ProjetGLA2017/src/dao | |
| Normalized Distance (avg/max per package) | 0,4 | 0,432 | | 1 | /ProjetGLA2017/src/metier/entities | |
| Number of Static Methods (avg/max per class) | 2 | 0,087 | 0,282 | 1 | /ProjetGLA2017/src/dao/TestDao.java | |
| Number of Interfaces (avg/max per package) | 4 | 1,333 | 1,886 | 4 | /ProjetGLA2017/src/dao | |
| Total Lines of Code | 1718 | | | | | |
| Weighted methods per Class (avg/max per class) | 264 | 11,478 | 12,072 | 53 | /ProjetGLA2017/src/web/ControleurServlet.java | |
| Number of Methods (avg/max per type) | 168 | 7,304 | 6,956 | 29 | /ProjetGLA2017/src/metier/entities/Client.java | |
| Depth of Inheritance Tree (avg/max per class) | 1,087 | 0,408 | | 3 | /ProjetGLA2017/src/web/ControleurServlet.java | |
| Number of Packages | 3 | | | | | |
| Instability (avg/max per package/Fragile) | 0,644 | 0,457 | | 1 | /ProjetGLA2017/src/web | |
| McCabe Cyclomatic Complexity (avg/nodes) | 1,553 | 3,871 | | 51 | /ProjetGLA2017/src/web/ControleurServlet.java | doGet |
| Nested Block Depth (avg/max per method) | 1,253 | 0,651 | | 4 | /ProjetGLA2017/src/web/ControleurServlet.java | doGet |
| Lack of Cohesion of Methods (avg/max per method) | 0,384 | 0,404 | 0,892 | 1 | /ProjetGLA2017/src/metier/entities/Client.java | |
| Method Lines of Code (avg/max per method) | 1029 | 6,053 | 27,108 | 345 | /ProjetGLA2017/src/web/ControleurServlet.java | doGet |
| Number of Overridden Methods (avg/parent) | 7 | 0,304 | 0,687 | 3 | /ProjetGLA2017/src/web/ControleurServlet.java | |
| Afferent Coupling (avg/max per package) | 6,667 | 8,731 | | 19 | /ProjetGLA2017/src/metier/entities | |
| Number of Children (avg/max per type) | 0 | 0 | 0 | 0 | /ProjetGLA2017/src/dao/PaymentTest.java | |

FIGURE 10 – Extrait du rapport de mesure de qualité

6 Conclusion générale

Notre projet de Génie logiciel avancé (GLA) a consisté au développement d'une application permettant de gérer le paiement des prestations au sein de la Résidence universitaire Ky tuc xa My Dinh. Dans le cadre de ce projet, notre intérêt s'est porté sur ce sujet en raison du besoin réel d'un tel système dans un écosystème tel que la Résidence universitaire dans lequel il y a plus de 50 000 étudiants effectuant des milliers de paiements par jour. Nous pouvons, en effet, mesurer la complexité de gestion et de contrôle d'un tel système avec d'un côté les étudiants et de l'autre côté les prestataires.

Cependant, nous tenons à souligner que notre démarche s'inscrit dans un cadre d'exploration afin d'avoir une idée claire du projet. Ainsi, les techniques de développement logiciel n'ont pas assez été développées dans le cadre du présent projet. En outre, ce projet constitue notre premier pas dans l'ingénierie logicielle dans un cas réel ce qui peut aussi justifier le manque de technique.

A l'issu de ce travail, nous pouvons dire que ce projet a répondu largement à nos attentes aussi bien au niveau professionnel que relationnel. Nous avons beaucoup gagné en compétences en ce qui concerne les technologies Java EE, le domaine bancaire et du paiement électronique et l'interfaçage avec le monde extérieur. En outre, nous avons implémenté les techniques de développement.

Difficultés rencontrées. Les difficultés majeures que nous avons rencontrées durant ce projet sont en autre :

- La définition claire et précise des fonctionnalités du système. En effet, le projet est partie d'une idée du Gestionnaire de la Résidence. Nous avons donc du travailler à détailler et à préciser les attentes du projet.
- Le développement sans framework. Sans framework, la charge de développement a été doublée. En effet, nous avons du coder manuellement les connexions à la base de données et les relations entre les classes.

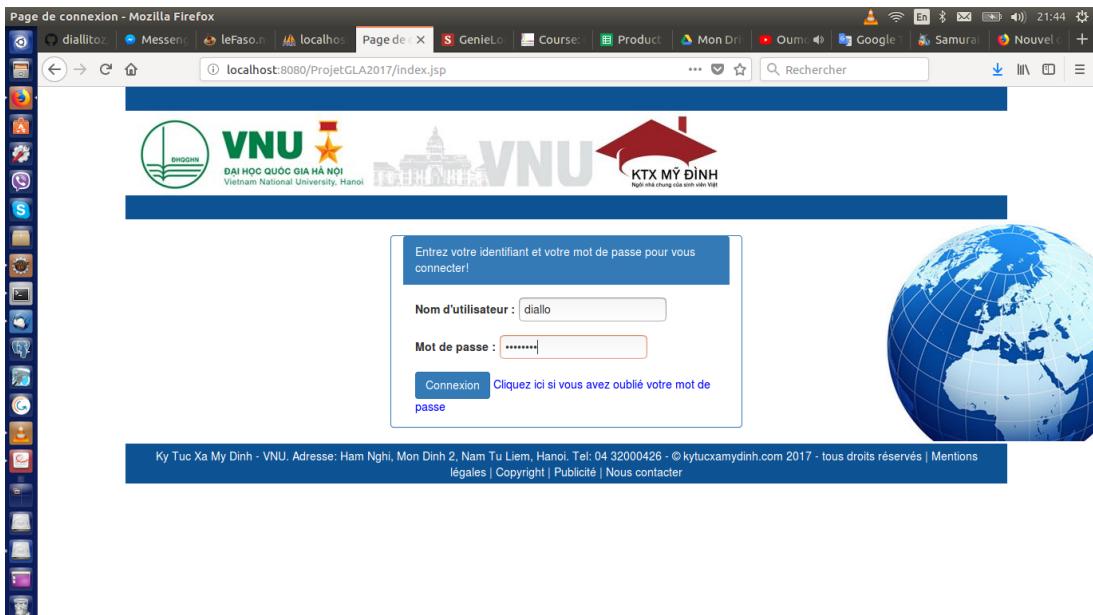


FIGURE 11 – Page d'accueil : Authentification

Perspectives. Plusieurs points s'avèrent intéressants à explorer en vue d'améliorer l'application.

- L'utilisation d'un framework tel que Spring pour la reprise du développement de l'application.
- La création des alertes (par email ou par messages téléphoniques) par rapport au mouvement sur les comptes.
- La possibilité d'utiliser d'autres technologies Web avancées telles que les services Web pour assurer l'interopérabilité de l'application avec d'autres systèmes dans un environnement hétérogène (gestion des étudiants, gestion comptable...).
- L'utilisation d'un lecteur magnétique de carte afin d'éviter à client de préciser son numéro ce compte pour effectuer des paiements.
- La création des graphes afin d'avoir des statistiques plus explicites.

7 Annexe : Présentation de l'application

7.1 Premier lancement de l'application

Ce premier lancement nous redirige vers une page d'authentification, un formulaire qui permet de renseigner le nom d'utilisateur ainsi que son mot de passe pour accéder à l'application (FIGURE 11).

Nous avons l'affichage de la page d'accueil correspondant au profil de l'utilisateur si toutes les informations fournies sont correctes, sinon le système renvoie la même page avec un message d'erreur (voir FIGURE 12).

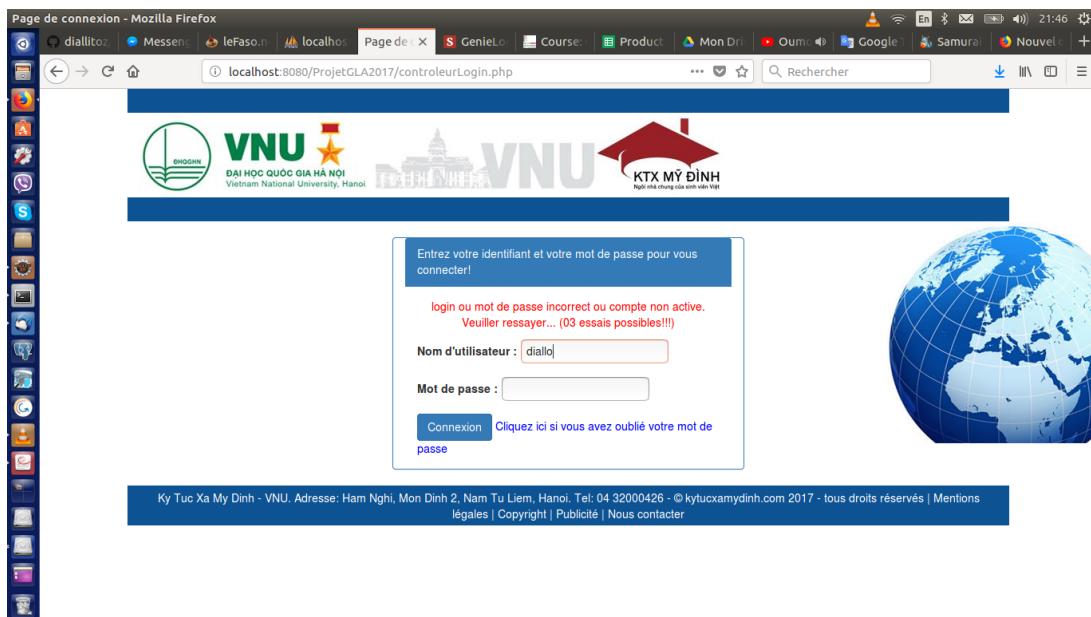


FIGURE 12 – Page d'accueil : Erreur de d'authentification

7.2 Accueil Administration

Un administrateur étant celui qui est chargé de gérer les utilisateurs ainsi que le système (voir FIGURE 13).

7.3 Accueil Gestionnaire

Un gestionnaire est chargé de gérer les clients et leurs comptes (voir FIGURE 14).

7.4 Accueil Étudiant

Un étudiant est un client. De ce fait, il peut consulter son solde, effectuer des paiements... (voir FIGURE 15).

7.5 Opérations sur les comptes client

Une fois connecté, un gestionnaire peut effectuer des opérations sur un compte client. Ainsi, après avoir choisi le compte (Menu Rechercher compte), il peut soit modifier, supprimer ou effectuer un dépôt (versement) (voir FIGURE 16).

7.6 Effectuer un versement (dépôt)

Après avoir rechercher le compte, cliquer sur *créditer* et remplir les informations (voir FIGURE 16).

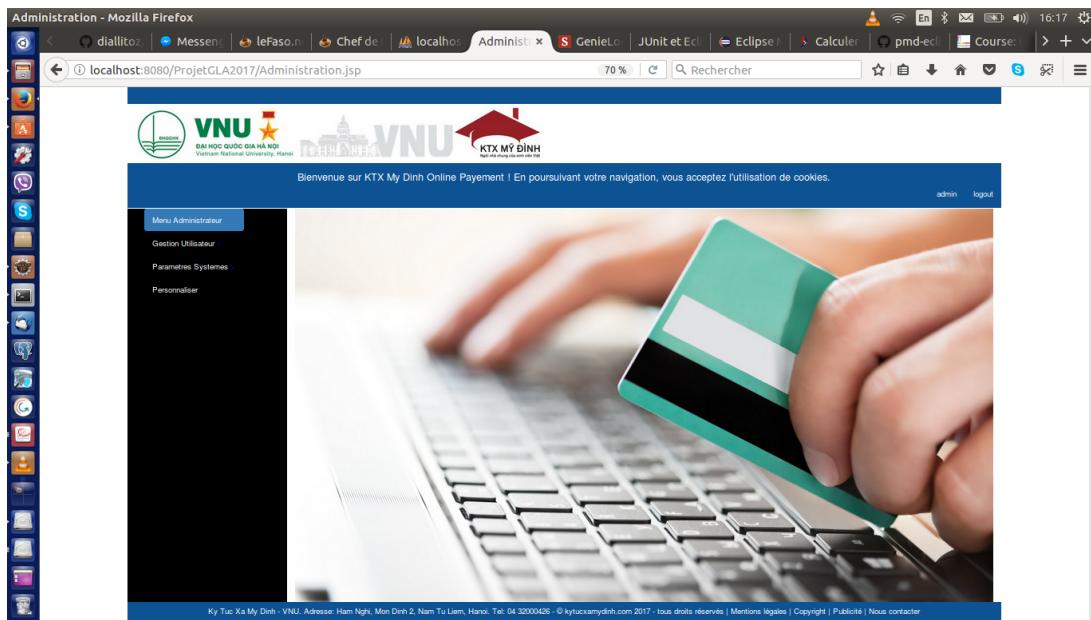


FIGURE 13 – Page d'accueil : Administration

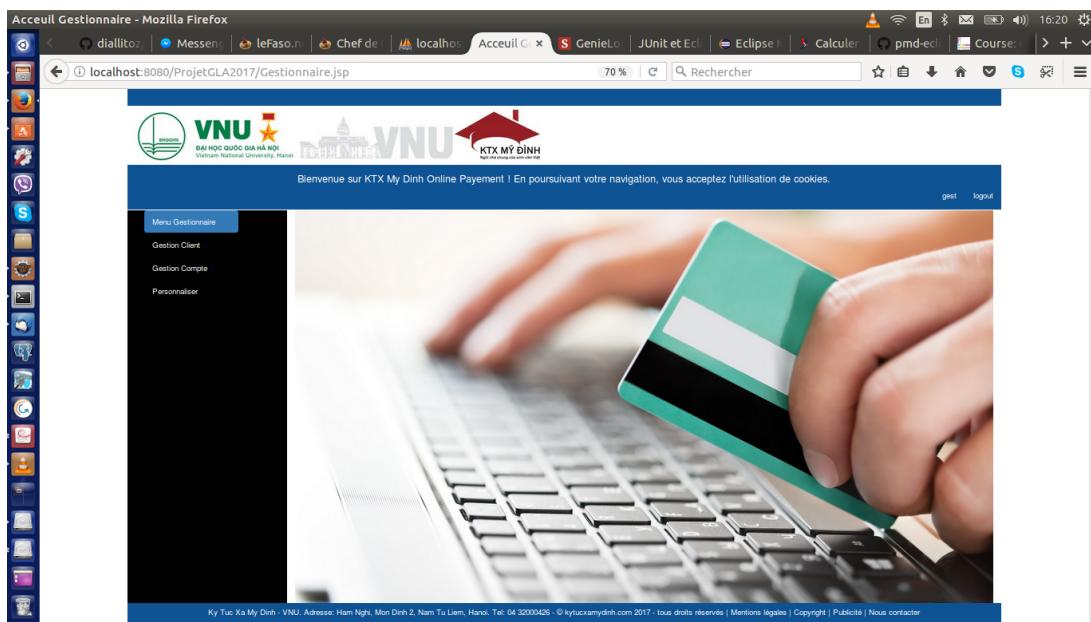


FIGURE 14 – Page d'accueil : Gestionnaire

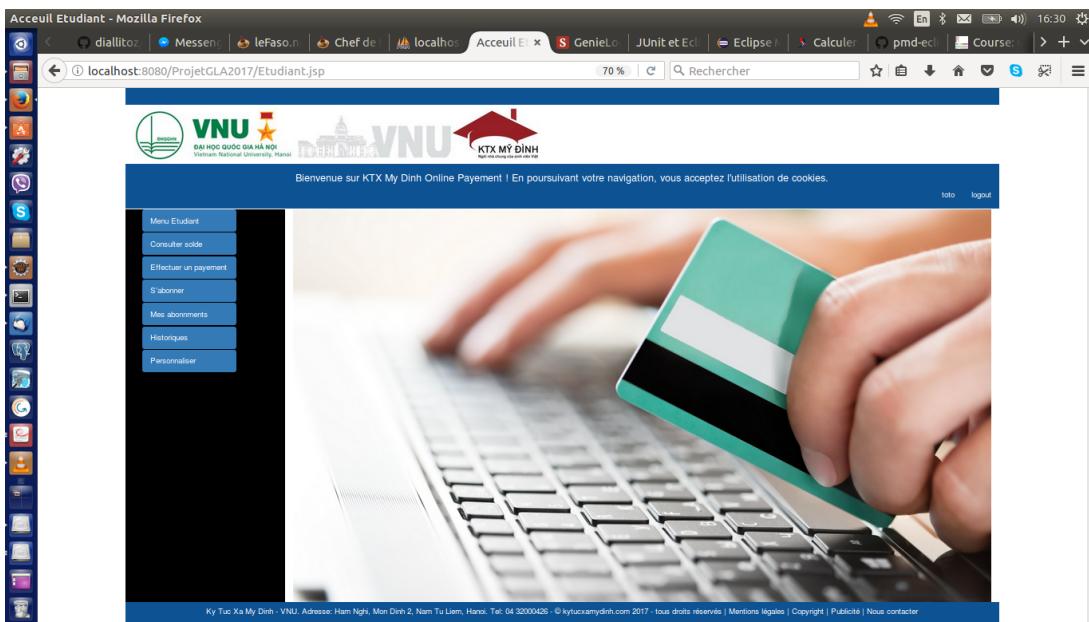


FIGURE 15 – Page d'accueil : Étudiant

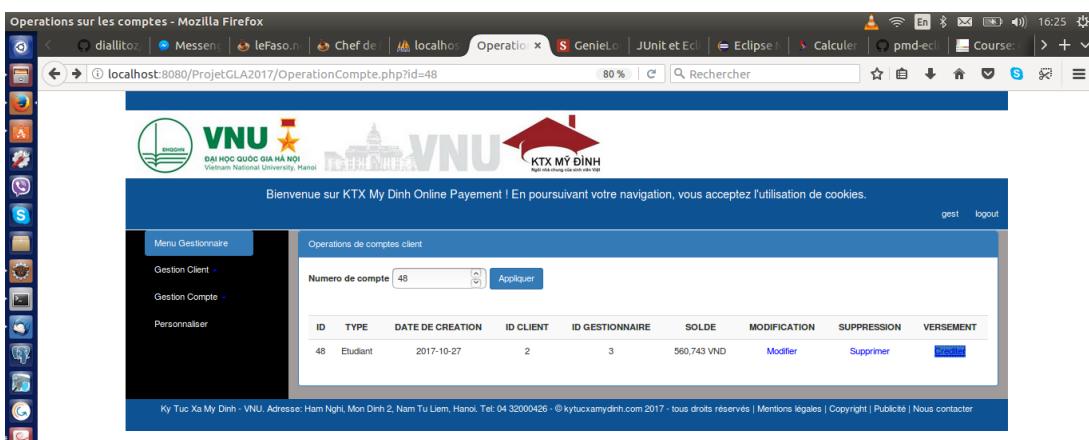


FIGURE 16 – Page des opérations d'un gestionnaire

FIGURE 17 – *Page de versement d’argent dans un compte*

FIGURE 18 – *Message de confirmation de versement*

7.7 Confirmation d’un versement (dépôt)

Si le versement s’est bien passé, le gestionnaire reçoit un message de confirmation (voir FIGURE 18).

7.8 Consulter solde

Un client peut consulter son solde, pour ce faire, il clique sur *Consulter solde* et remplir les informations (voir FIGURE 19).

7.9 Effectuer un payement

Un client peut effectuer un payement, pour ce faire, il clique sur *Effectuer un payement* et remplir les informations (voir FIGURE 20).

Bienvenue sur KTX My Dinh Online Payment ! En poursuivant votre navigation, vous acceptez l'utilisation de cookies.

| MON ID | TYPE | DATE | ID GESTIONNAIRE | SOLDE |
|--------|----------|------------------------------|-----------------|---------------|
| 2 | Etudiant | Fri Nov 17 16:30:52 ICT 2017 | 3 | 1,560,743 VND |

FIGURE 19 – Page de consultation du solde

Bienvenue sur KTX My Dinh Online Payment ! En poursuivant votre navigation, vous acceptez l'utilisation de cookies.

Date de paiement : Sat Nov 18 22:53:10 ICT 2017

Numero Compte du payeur : 48

Numero Compte du bénéficiaire : 49

Montant à payer* : 1000000

Type de paiement* : Boutique

Payer Renitialiser

FIGURE 20 – Page de paiement

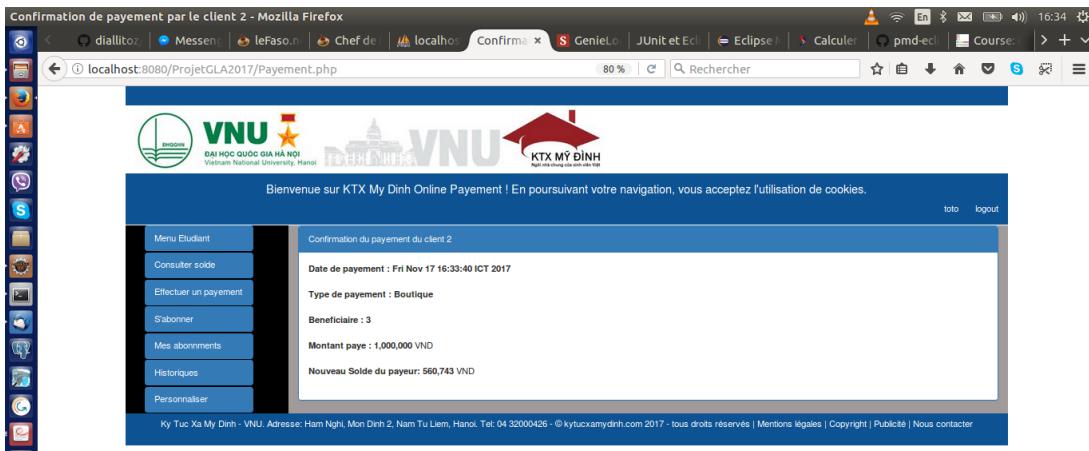


FIGURE 21 – Message de confirmation de paiement

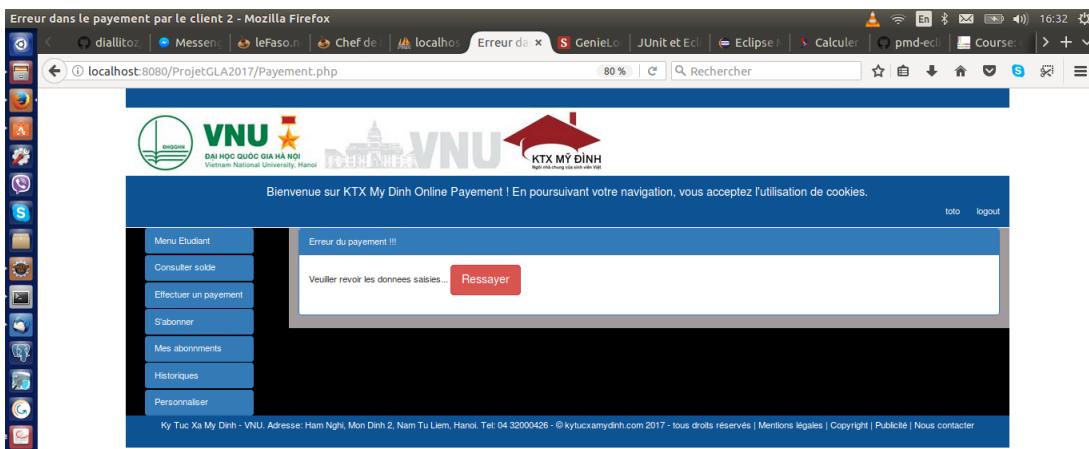


FIGURE 22 – Message d'erreur lors d'un paiement

7.10 Confirmation de paiement

Si le paiement s'est bien passé, le payeur reçoit un message de confirmation (voir FIGURE 21). Dans le cas contraire, un message d'erreur lui ait retourné (voir FIGURE 22).

7.11 Historiques des paiements reçus par un prestataire

Un prestataire peut consulter les paiements effectuées pour son compte. Pour ce faire, il clique sur *Historiques* (voir FIGURE 23).

Références

- [1] Draw.io. Draw.io. <https://www.draw.io/>, consultée le 01 octobre 2017.
- [2] Stève SFARTZ. Modèle multicouche Architecture à 5 couches : société IMPROVE.
- [3] Wikipedia. Github — wikipédia. <https://fr.wikipedia.org/wiki/GitHub>, 15 novembre 2017 à 14:43., consultée le 30 septembre 2017.

| ID | TYPE | MONTANT | COMPTE PAYEUR | DATE | IMPRIMER | EXPORTER |
|----|----------|----------|---------------|------------------------------|--------------------------|--------------------------|
| 3 | Boutique | 49.0 | 48 | Tue Nov 07 22:11:50 ICT 2017 | Imprimer | Exporter |
| 4 | Boutique | 49.0 | 48 | Tue Nov 07 22:15:29 ICT 2017 | Imprimer | Exporter |
| 9 | Boutique | 55555.0 | 48 | Wed Nov 08 11:56:07 ICT 2017 | Imprimer | Exporter |
| 11 | Boutique | 100000.0 | 48 | Wed Nov 08 13:28:40 ICT 2017 | Imprimer | Exporter |
| 15 | Boutique | 100000.0 | 48 | Thu Nov 09 14:01:24 ICT 2017 | Imprimer | Exporter |
| 24 | Boutique | 100000.0 | 48 | Fri Nov 17 16:33:40 ICT 2017 | Imprimer | Exporter |

FIGURE 23 – *Historiques des paiements reçus par un prestataire*

- [4] Wikipedia. Modèle-vue-contrôleur — wikipédia. <https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur>, 20 octobre 2017 à 04 :42., consultée le 01 octobre 2017.
- [5] Wikipedia. Sharelatex — wikipédia. <https://fr.wikipedia.org/wiki/ShareLaTeX>, 29 mai 2017 à 17 :11., consultée le 01 octobre 2017.