

Université de Mons
Faculté Des Sciences
Département d’Informatique

**Projet de modélisation logicielle
Rapport de modélisation**

Professeur :

Tom MENS

Assistants :

Jeremy DUBRULLE

Gauvin DEVILLEZ

Sébastien BONTE

Auteurs :

Pignozi AGBENDA

Thomas BERNARD

Théo GODIN

Ugo PROIETTI



Année académique 2021-2022

Projet de Modélisation Logicielle

AGBENDA Christian, BERNARD Thomas, GODIN Theo et PROIETTI Ugo

Table des matières

1	Introduction	1
2	Applications de bases	1
2.1	Application1 : Gestion de portefeuilles financiers	1
2.1.1	Vue d'ensemble	1
2.1.2	Diagramme de cas d'utilisation	2
2.1.3	Interaction Overview Diagram	2
2.1.4	Diagramme de classes	3
2.1.5	Diagrammes de séquences	5
2.2	Application 2 : Application de gestion pour institutions financières	12
2.2.1	Vue d'ensemble	12
2.2.2	Diagramme de cas d'utilisation	12
2.2.3	Interaction Overview Diagram	13
2.2.4	Diagramme de classes	14
2.2.5	Diagramme de séquences	16
2.3	Serveur	20
2.3.1	Vue d'ensemble	20
2.3.2	Diagramme d'entité-relation	20
2.3.3	API	23
2.4	Interface Graphique	26
2.4.1	Application 1	26
2.4.2	Application 2	37
2.4.3	Application clients	37
3	Extensions	49
3.1	Extension 1, Gestion des cartes - Godin Théo :	49
3.1.1	Application 1	49
3.1.2	Use cases de l'application 1	49
3.1.3	Interaction overview de l'application 1	49
3.1.4	Diagramme de classes de l'application 1	49
3.1.5	Diagrammes de séquences de l'application 1	50
3.1.6	Interface graphique de l'application 1	50
3.1.7	Application 2	54
3.1.8	Use cases de l'application 2	54
3.1.9	Interaction overview de l'application 2	55

3.1.10	Diagramme de classes de l'application 2	55
3.1.11	Interface graphique de l'application 2	55
3.1.12	Diagramme d'entité relation	55
3.2	Extension 2, Gestion des devises et virements internationaux - Proietti Ugo :	57
3.2.1	Diagramme des cas d'utilisation de l'application 1	57
3.2.2	Diagramme de séquence	58
3.2.3	Classes de l'application 1	58
3.2.4	Classes de l'application 2	59
3.2.5	Interface graphique de l'application 1	60
3.2.6	Interface graphique de l'application 2	61
3.2.7	Diagramme de l'API (financial product)	61
3.2.8	Base de données	62
3.2.9	Interaction Overview Diagram 1	63
3.2.10	Interaction Overview Diagram 2	64
3.3	Extension 5, Gestion des contrats d'assurance - Bernard Thomas :	65
3.3.1	Vue d'ensemble	65
3.3.2	Application 1	65
3.3.3	Diagramme des cas d'utilisation	65
3.3.4	Interaction Overview Diagram	67
3.3.5	Diagrammes de classe	68
3.3.6	Diagrammes de séquence	69
3.3.7	Application 2	74
3.3.8	Diagramme des cas d'utilisitation	74
3.3.9	Interaction Overview Diagram	74
3.3.10	Diagramme de classes	74
3.3.11	Diagrammes de séquences	77
3.3.12	Serveur	77
3.3.13	Diagramme d'entité relation	77
3.3.14	Interface graphique de l'extension	78
3.3.15	Application 1	78
3.3.16	Application 2	84
3.4	Extension 6, Paiement et gestion des fraudes - Agbenda Pignozi :	86
3.4.1	Vue d'ensemble	86
3.4.2	Application 1	86
3.4.3	Diagramme des cas d'utilisation	86
3.4.4	Interaction Overview Diagram	87
3.4.5	Diagrammes de classe	88
3.4.6	Diagrammes de séquence	88
3.4.7	Application 2	90
3.4.8	Interaction Overview Diagram	90
3.4.9	Diagramme de classes	90
3.4.10	Diagrammes de séquences	91
3.4.11	Serveur	92
3.4.12	Diagramme d'entité relation	92

1 Introduction

Ce rapport recense tous les diagrammes UML composant nos applications de base ainsi que les 4 extensions de notre groupe. Ce rapport contient également le diagramme d'API ainsi que les diagrammes d'entité-relation général et ses versions étendues pour chacune de nos extensions. Le projet est composé de 2 applications : L'application 1 qui est une application de gestions de portefeuilles d'un point de vue client et l'application 2 qui est l'application de gestion du point de vue institutionnel. Ces deux applications ont une modélisation assez semblable en terme d'architecture mais diffèrent l'une de l'autre du point de vue de leurs fonctionnalités.

Notre phase de modélisation a parfois été semée d'embuches. Tantôt dans la compréhension des consignes tantôt dans la découverte de certains concepts informatiques qui nous étaient encore inconnus jusque là. Nous avons toutefois su faire nos recherche afin d'amener notre compréhension à un niveau suffisant pour nous permettre de manipuler non sans difficulté et maladresse ces nouveaux concepts.

Cette modélisation nous a permis de mieux nous projeter dans ce que sera l'implémentation du projet.

2 Applications de bases

2.1 Application1 : Gestion de portefeuilles financiers

2.1.1 Vue d'ensemble

L'application 1 est l'application qui est destinée aux clients. Il s'agit de l'application sur laquelle nous avons commencé à travailler et donc celle qui a subi le plus de modifications au total. Il est possible que certains concepts soient modifiés lors de l'implémentation notamment ceux qui n'ont pas été approfondis au maximum. Cette application sera étendue par l'entièreté du groupe.

2.1.2 Diagramme de cas d'utilisation

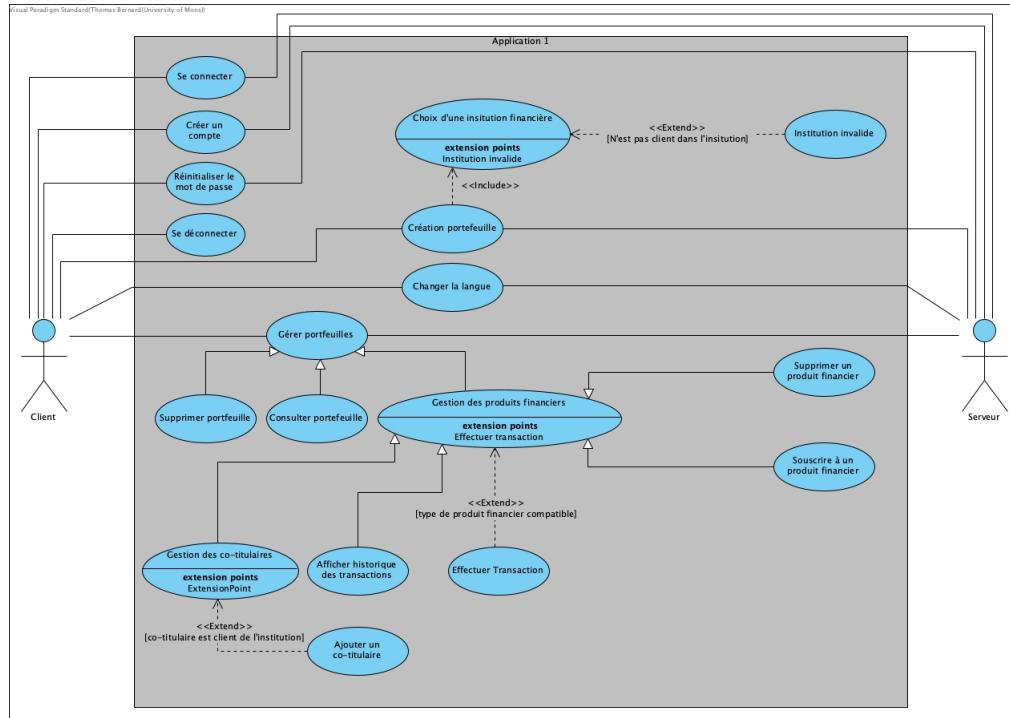


FIGURE 1 – Diagramme des cas d'utilisation de l'Application 1.

Pour le diagramme de cas d'utilisation nous n'avons pas rencontrés beaucoup de difficultés. Il nous a suffit de lire les consignes du projet et d'en ressortir les points importants qui devaient être les fonctionnalités principales de notre projet. Ce diagramme nous a servi de squelette dans la création de la suite de nos diagrammes. Il n'y a pas remarques importantes à faire à ce sujet car le diagramme ne fait que recenser les actions basiques de l'application 1.

2.1.3 Interaction Overview Diagram

Le schéma des interactions de l'application utilisateurs est composée de 3 parties principales. Une partie dédiée à l'accès à l'application (authentification), une partie pour la l'accès et la gestion des portefeuilles financiers la dernière pour l'accès et la gestion des produits financiers.

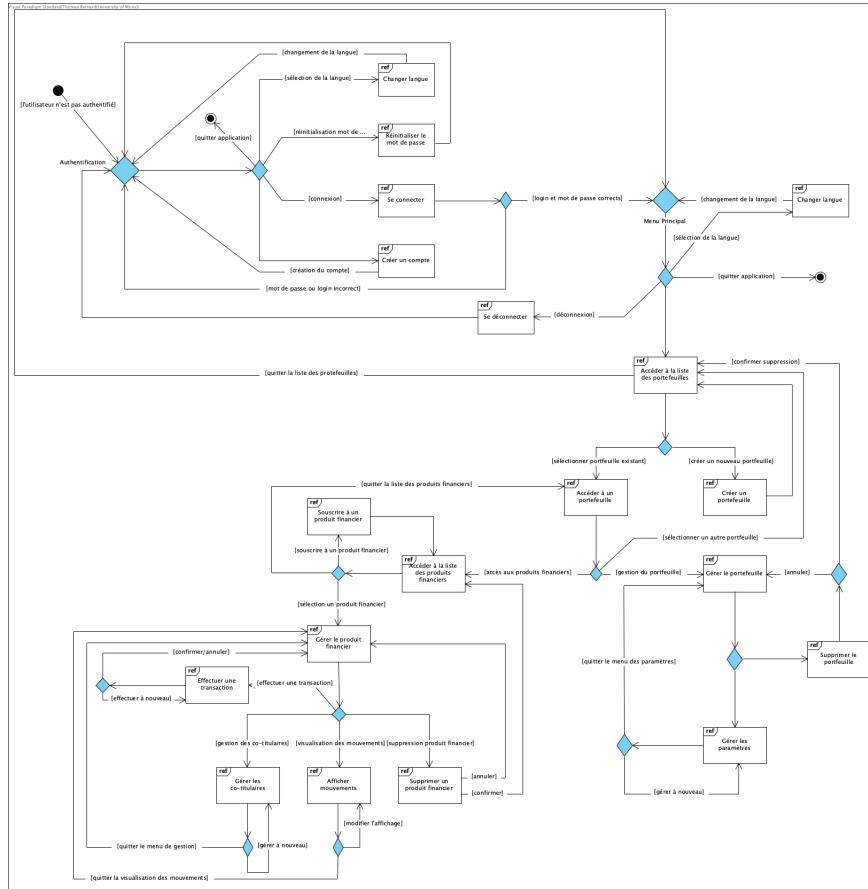


FIGURE 2 – Interaction Overview Diagram Application 1.

2.1.4 Diagramme de classes

Les classes de l'application 1 ont été divisées en 3 packages afin de garder une bonne organisation générale des classes. Il y a donc un package APP qui contient la partie logique de l'application comme la représentation des objets, un package GUI contenant toutes les classes de l'interface graphique et un package API contenant toutes les classes permettant la communication avec le serveur et donc la base de données.

Pour la partie logique, chaque objet possède une classe Data permettant de créer un fichier JSON contenant les données de cet objet. Cela permettra de facilement envoyer des données au serveur via l'API.

La partie graphique de l'application est un ensemble de scènes créées à partir d'une scène de base. Chaque scène est spécifique à une fenêtre de l'application.

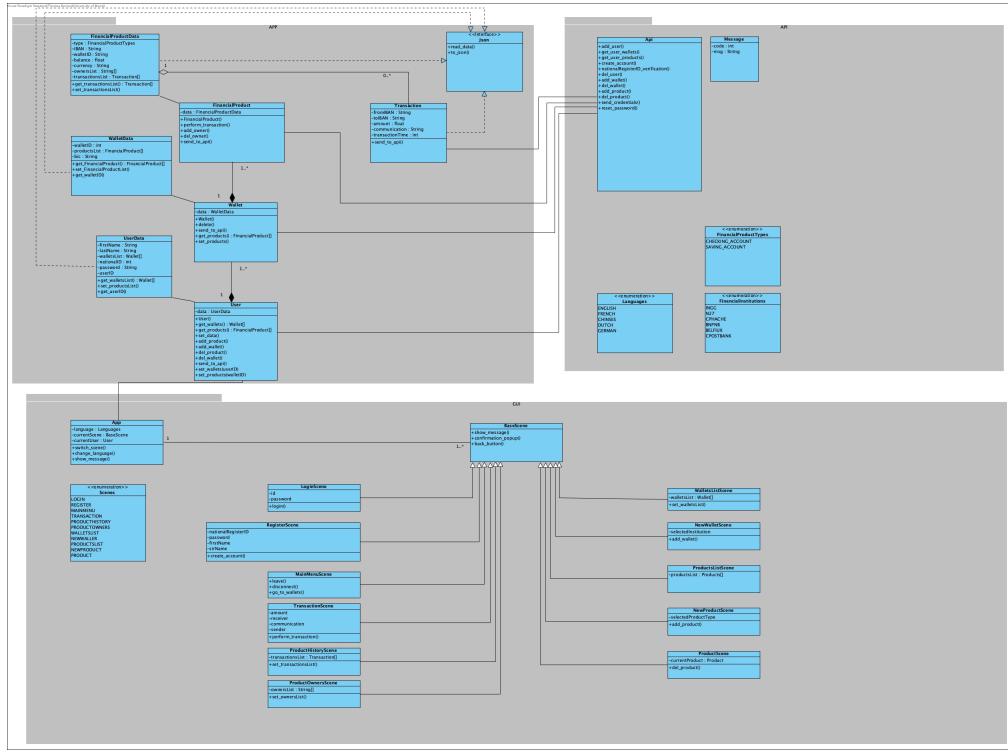


FIGURE 3 – Diagramme de classes de l’application 1

La classe App du package GUI permettra de passer d’une scène à une autre. La navigation entre les scènes sera facilitée par l’enumeration Scenes contenant une énumération par scène. App possède également une référence vers un objet currentUser afin de pouvoir accéder aux données de l’utilisateur courant.

Les méthodes de l’API se résumeront globalement à l’envoi d’un message via HTTPS sur l’URI de l’API. Les paramètres sont passés directement dans l’URI ce qui posera des problèmes de sécurité lors de l’implémentation. (explication dans la partie API du rapport)

Pour passer des objets à l’API, nous utiliserons JSON ou YAML. Elle répond avec le même format. JSON ou YAML sont très simples à générer et à lire d’où le grand intérêt de leur utilisation. D’autres méthodes vont être nécessaires afin d’interagir avec les fichiers JSON ou YAML.

2.1.5 Diagrammes de séquences

1. Accéder à la liste des portefeuilles

Ce diagramme de séquence décrit comment l'application va récupérer

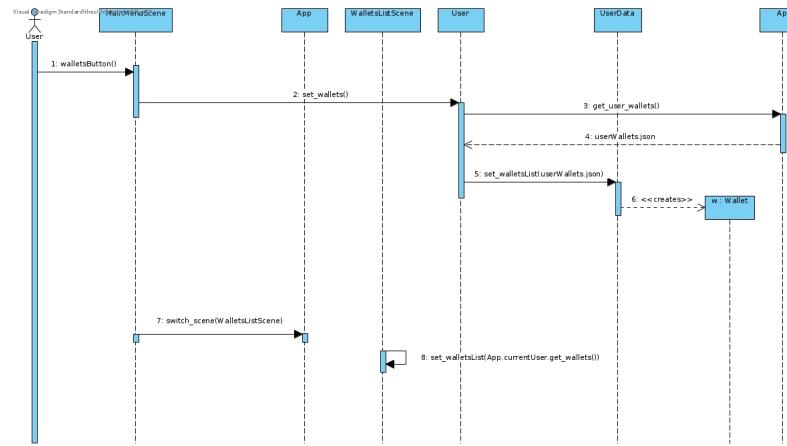


FIGURE 4 – Accéder à la liste des portefeuilles

la liste des portefeuilles de l'utilisateur et la stocker au sein de l'application. L'application va ensuite changer de scène pour passer sur l'écran d'affichage des portefeuilles. Cette scène pourra alors accéder facilement à la liste des portefeuilles afin de les afficher à l'utilisateur.

2. Accéder à la liste des produits financiers

Lorsque l'utilisateur sélectionne un de ses portefeuilles, les produits fi-

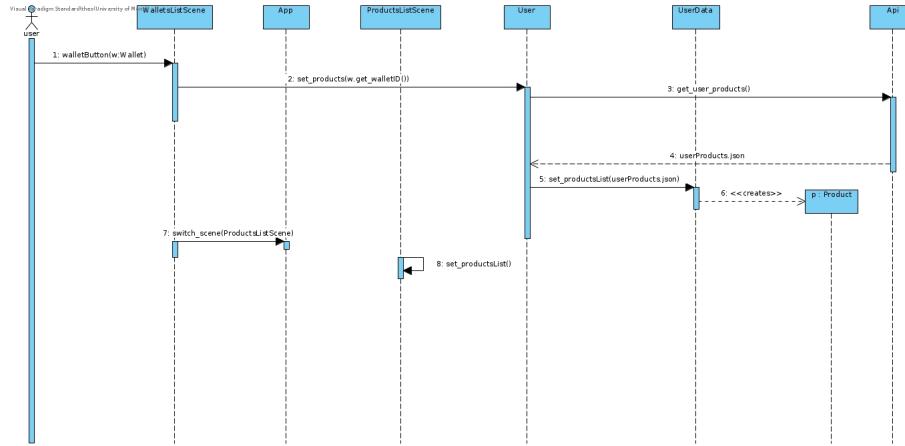


FIGURE 5 – Accéder à la liste des produits financiers

nanciers de ce portefeuilles doivent être récupérés depuis le serveur. Ils sont ensuite stockés dans une liste de l'objet Wallet. Une fois les produits récupérés, la scène des détails d'un portefeuille est affichée.

3. Créer un compte

Afin de créer un compte, l'utilisateur va entrer ses données qui seront

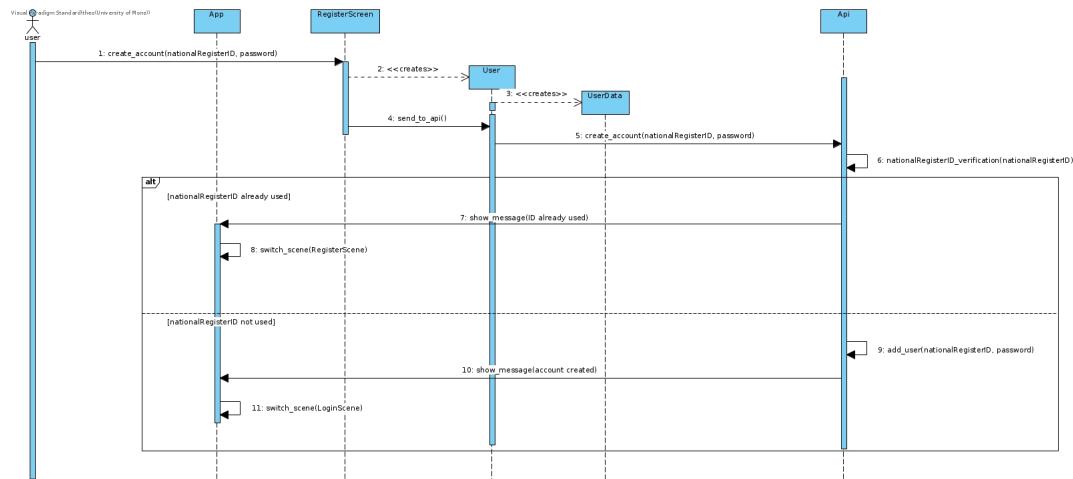


FIGURE 6 – Crée un compte

ensuite utilisées afin de créer un objet UserData. Cet objet va ensuite être envoyé vers le serveur sous forme d'un fichier json.

4. Créer un portefeuille

La création d'un portefeuille demande peu d'informations à l'utilisateur.

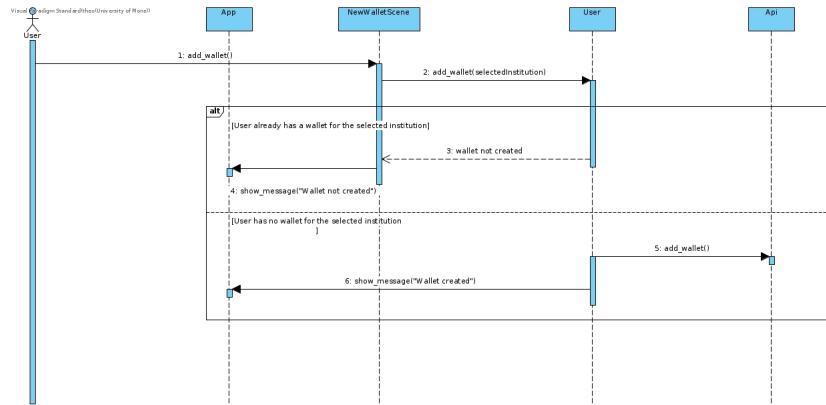


FIGURE 7 – Créer un portefeuille

Une fois qu'il a sélectionné l'institution pour laquelle il veut créer un portefeuille, l'application va vérifier si il a déjà un portefeuille pour cette institution. Si c'est le cas, une notification d'erreur lui sera affichée. Sinon le portefeuille sera créé et l'information sera envoyée vers le serveur.

5. Effectuer une transaction

Une fois que l'utilisateur a entré les informations de la transaction et

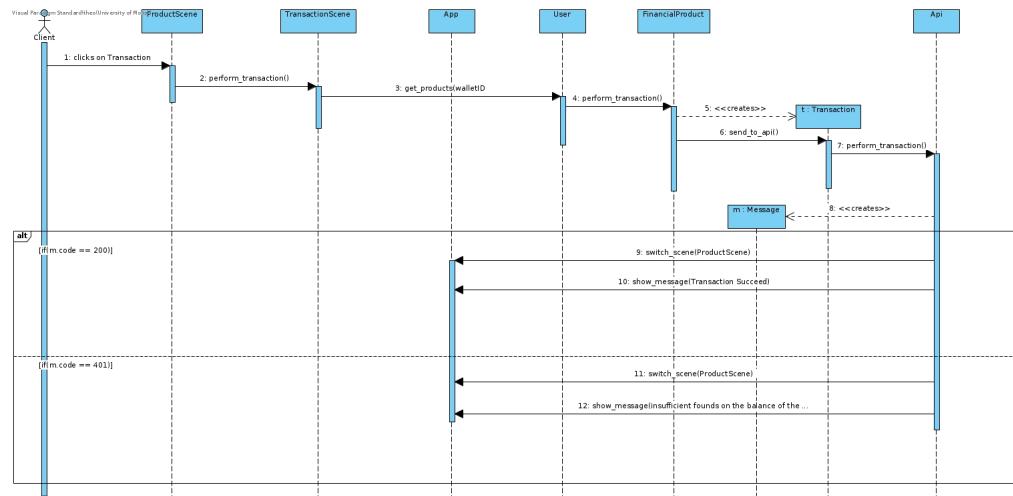


FIGURE 8 – Effectuer une transaction

accepté la confirmation, un objet Transaction est créé et envoyé vers le serveur.

6. Se connecter

La procédure de connexion à l'app est initiée lorsque l'utilisateur entre

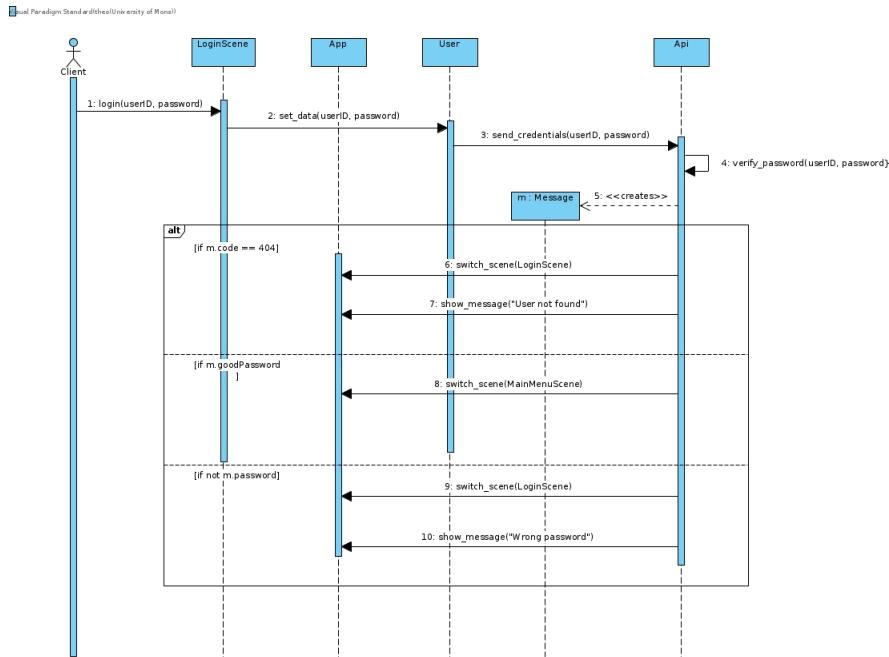


FIGURE 9 – Se connecter

ses identifiants et clique sur se connecter. Ses identifiants seront alors envoyés à l'api qui vérifiera ceux-ci. Si ils sont corrects alors l'utilisateur est envoyé vers le menu principal de l'application. Si le nom d'utilisateur est incorrect un message “user not found” lui est affiché et si le mot de passe est incorrect alors un message “wrong password” est affiché.

7. Souscrire à un produit financier

Lorsqu'un produit financé est créé par l'utilisateur, cela envoie un objet

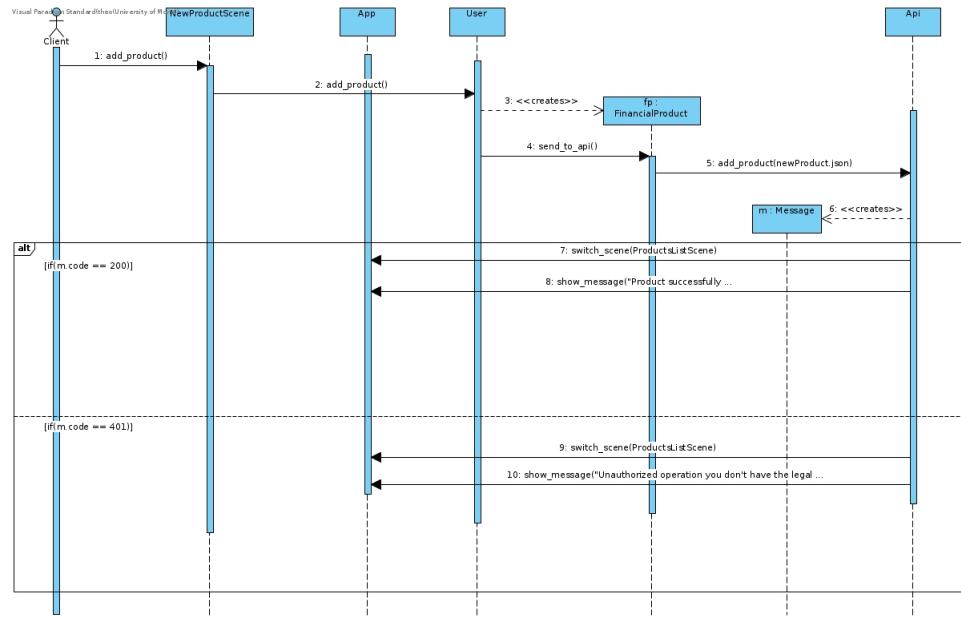


FIGURE 10 – Souscrire à un produit financier

FinancialProduct à l'api qui se chargera de l'envoyer vers le serveur. Un message est ensuite envoyé à l'utilisateur afin de l'informer de la réussite (ou échec) de l'opération.

2.2 Application 2 : Application de gestion pour institutions financières

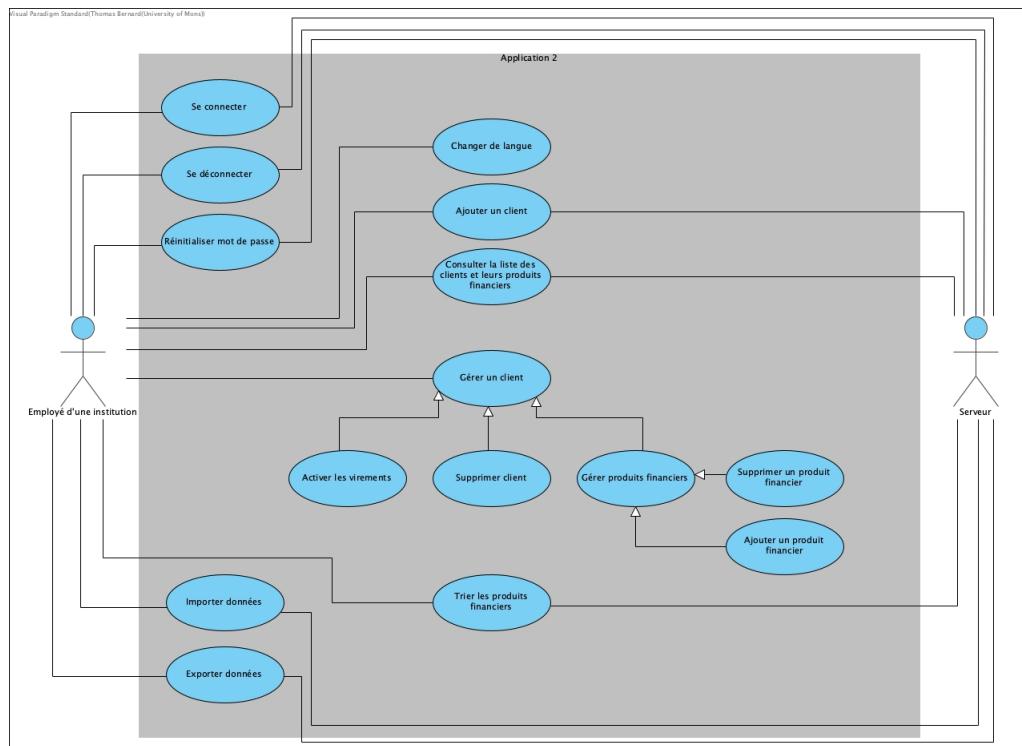
2.2.1 Vue d'ensemble

L'application 2 est l'application destinée aux institutions. Cette application leur permet de gérer leurs clients ainsi que les produits de ceux-ci. La conception de cette application est basée sur celle de l'application 1. En effet l'application 2 reprend certains des aspects de l'application 1. Tout en étant plus restrictive. En effet, l'application 2 n'a pas d'accès à la notion de portefeuille. Toutefois, elle possède des outils de gestion qui ne font pas partie de l'application 1.

2.2.2 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation a été assez simple à réaliser car il a suffit de se baser sur l'application 1 et de bien comprendre les consignes. Il constitue la structure de l'application pour les institutions.

Le diagramme de cas d'utilisation 2 partage plusieurs points communs avec le diagramme de cas d'utilisation 1 ce qui facilitera l'implémentation par la suite. De plus, il est moins complexe et moins lourd que ce dernier.



2.2.3 Interaction Overview Diagram

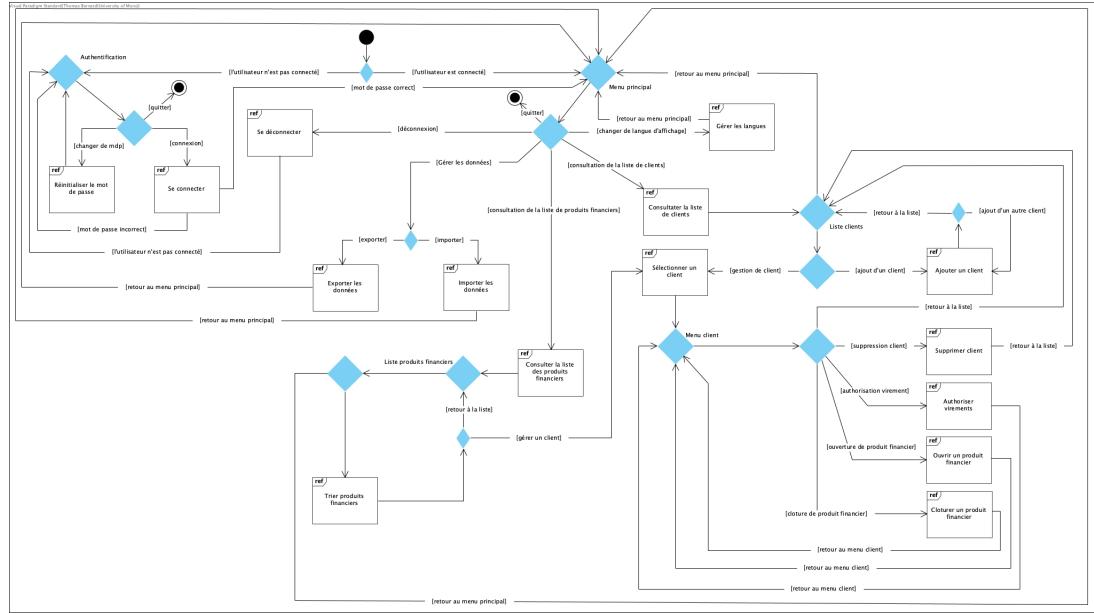


FIGURE 11 – Interaction Overview Diagram Application 2.

L'utilisateur d'un application d'institution financière devra d'abord s'authentifié via le menu d'authentification schématisé dans le coin supérieur gauche du diagramme.

Il devra utiliser le numéro swift et mot de passe de l'institution pour laquelle il travaille. Il pourra ensuite choisir de rester connecté à l'application.

Le menu principal de l'application permet d'accéder à la liste des clients et ainsi depouvoir effectuer des opérations sur chaque client de l'institution ou d'en ajouter de nouveaux.

La partie gestion de clients est schématisée dans la partie droite du diagramme.

2.2.4 Diagramme de classes

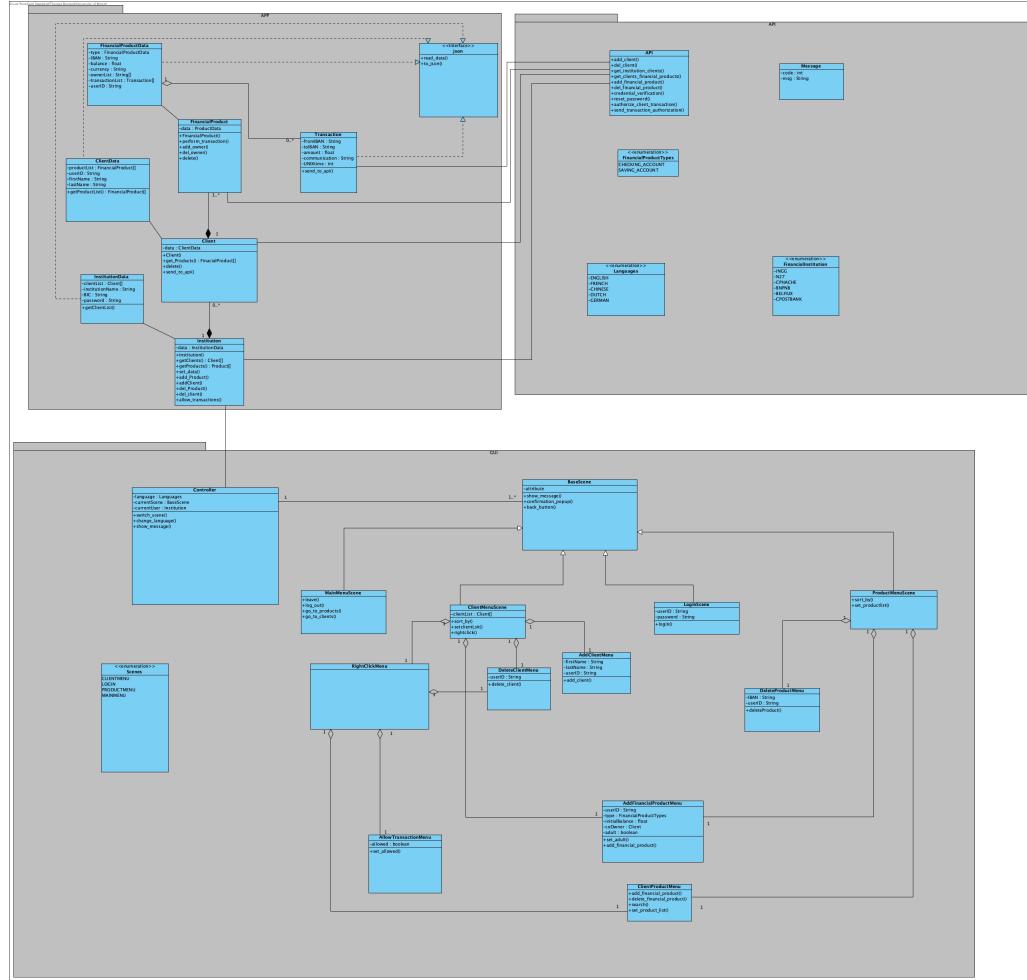


FIGURE 12 – Diagramme de classe de l’application 2

Les classes de l’application 2 ont également été divisées en 3 packages afin de garder une bonne organisation générale des classes. Il y a donc un package APP qui contient la partie logique de l’application comme la représentation des objets, un package GUI contenant toutes les classes de l’interface graphique et un package API contenant toutes les classes permettant la communication avec le serveur et donc la base de données.

Pour la partie logique, chaque objet possède une classe Data permettant de créer un fichier JSON contenant les données de cet objet. Cela permettra de facilement envoyer des données au serveur via l’API.

La partie graphique de l'application est un ensemble de scènes créées à partir d'une scène de base avec des méthodes globaux avec pour la plus part leurs menus qui eux contiennent des méthodes plus spécifiques à une tâche en particulier. Chaque scène est spécifique à une fenêtre de l'application et chaque menu est spécifique à une action pouvant être effectuée dans la scène.

La classe Controller du package GUI permettra de passer d'une scène à une autre. La navigation entre les scènes sera facilitée par l'enumeration Scenes contenant une énumération par scène. L'App possède également une référence vers un objet currentInstitution afin de pouvoir accéder aux données de l'institution courante.

Les méthodes de l'API se résumeront globalement à l'envoi d'un message via HTTPS sur l'URI de l'API. Les paramètres sont passés directement dans l'URI ce qui posera des problèmes de sécurité lors de l'implémentation. (explication dans la partie API du rapport)

Pour passer des objets à l'API, nous utiliserons JSON ou YAML. Elle répond avec le même format. JSON ou YAML sont très simples à générer et à lire d'où le grand intérêt de leur utilisation. D'autres méthodes vont être nécessaires afin d'intégrer avec les fichiers JSON ou YAML.

2.2.5 Diagramme de séquences

1. Autoriser les virements

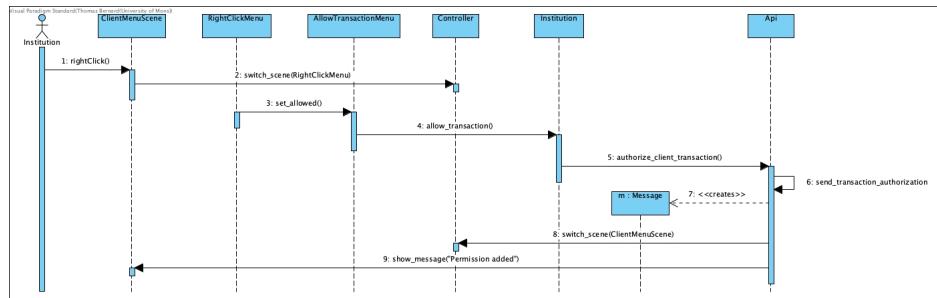


FIGURE 13 – Autoriser les virements

Ce diagramme décrit la procédure qu’effectue un employé d’une institution afin d’activer les virements pour un client de l’institution. La procédure est relativement simple, une fois que l’employée a cliqué sur le bouton d’activation, une requête est envoyée à l’api afin d’activer la fonctionnalité dans la base de données.

2. Cloturer un produit financier

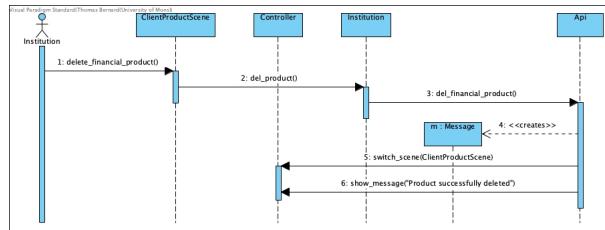


FIGURE 14 – Cloturer un produit financier

Lorqu'un employé d'institution cloture un produit financier d'un client, une requête est envoyée au serveur afin de désactiver le produit dans la base de donnée. Le produit n'est pas réellement définitivement supprimé.

3. Consulter la liste des produits financiers

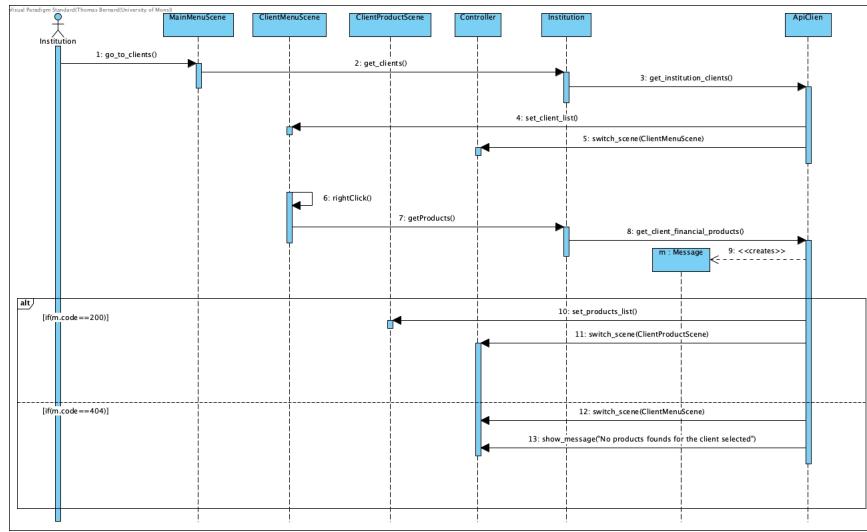


FIGURE 15 – Consulter la liste des produit financiers

Après avoir sélectionner un client, un employé d'une institution financière peut consulter la liste des produits de son client. Une première requête à l'api récupérera la liste des clients et une deuxième requête s'occupera de récupérer les produits d'un client sélectionné.

4. Supprimer un client

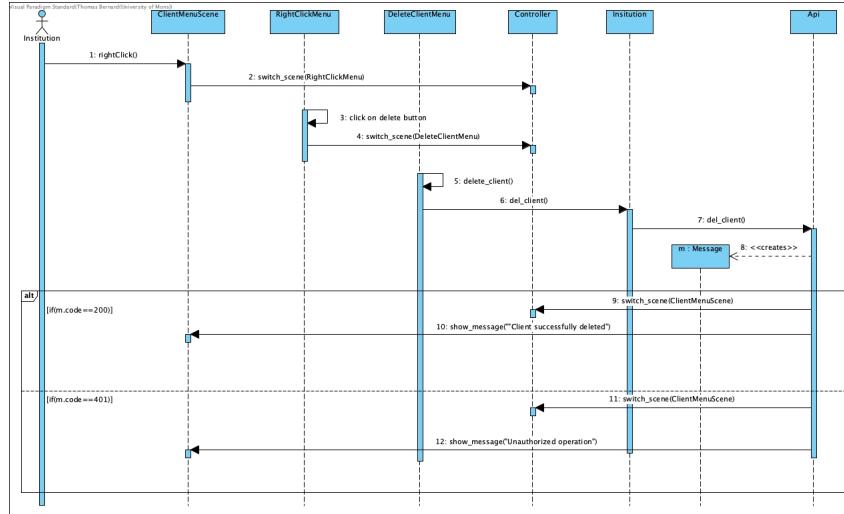


FIGURE 16 – Supprimer un client de l'institution

Un employé d'une institution peut choisir de supprimer un client via le menu de suppression de clients. Une requête est alors envoyée à l'api qui confirmera ensuite l'opération en indiquant si elle a réussi ou non.

2.3 Serveur

2.3.1 Vue d'ensemble

On pourrait croire que le serveur n'a pas une place importante dans notre modélisation car il n'est pas beaucoup représenté. Pourtant il a joué un rôle crucial dans la reflexion du projet car il est un peu l'élément central qui définit la logique des applications.

Ces dernières communiquent avec lui via une API de type REST qui sera implémentée en Java et hébergée sur AlwaysData. Nous retrouvons également la base de données représentée par l'Entity Relation Diagram sur le serveur.

Pour des raisons évidentes de sécurité, le serveur est le seul à pouvoir gérer la base de temps. Cela empêche tout problème du à un pc n'étant pas à l'heure et qui pourrait fausser l'heure d'exécution d'une transaction. Nous utilisons le temps au format UNIX time qui est stockable dans un simple int facilitant grandement les éventuelles opérations à faire.

2.3.2 Diagramme d'entité-relation

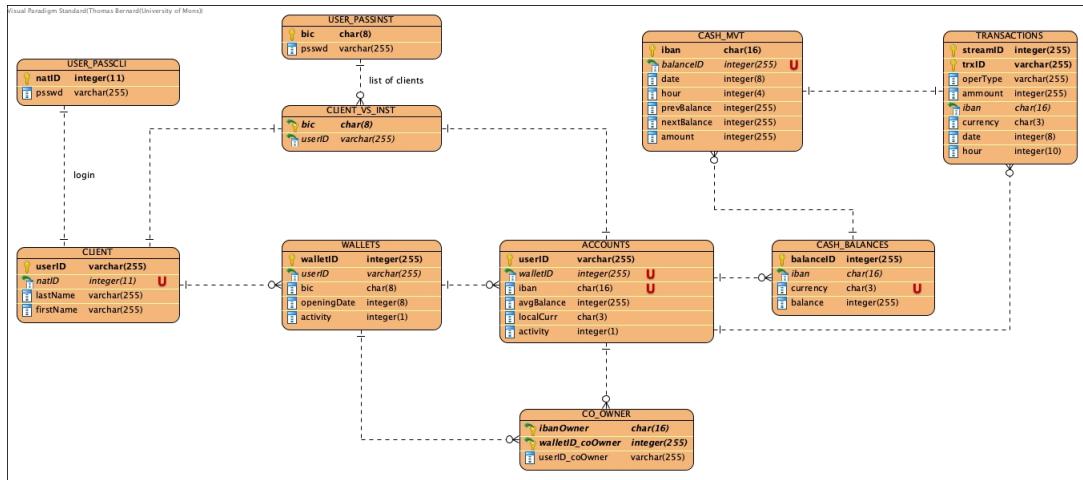


FIGURE 17 – Diagramme d'entité-relation de l'application.

Tables utilisateurs : Pour notre base de données nous avons essayé de séparer au mieux les tables afin que celles-ci restent simples de compréhension et faciles d'accès. Nous avons une table **USER_PASSCLI** qui sert de table d'authentification. Elle contient le mot de passe **psswd** ainsi que le numéro de registre national de l'utilisateur **natID** si les deux correspondent alors on peut accéder à la table **CLIENT** qui contient quant à les informations personnelles du client ainsi que son nom d'utilisateur **userID**.

Tables des institutions : Afin que les institutions n'aient pas accès à la notion de portefeuilles nous avons introduit 2 tables qui leurs sont propres et qui permettent de faire le lien entre les clients de la table **CLIENT** et leurs produits financiers de la table **ACCOUNTS**.

La table **USER_PASSINST** est à l'image de la table **USER_PASSCLI** une table d'authentification pour les institutions à l'aide de leur identifiant **bic**.

La table **CLIENT_VS_INST** associe chaque client et institution si pour un **userID** il possède au moins un produit financier dans l'institution. Lorsqu'un *wallet* est créé le **userID** du client ainsi que le **bic** de l'institution dans lequel il est créé sont ajoutés à la table ce qui permet ainsi à l'institution de récupérer la liste de ses clients assez aisément à l'aide de cette table. De plus elle peut ensuite se servir du **userID** afin de récupérer la liste des produits du client (table **ACCOUNTS** ainsi que ses informations personnelles (table **CLIENT**).

Autres tables La table **WALLETS** est la table qui contient tous les portefeuilles de clients. Lorsqu'un portefeuille est créé en plus d'ajouter le portefeuille dans la table **WALLETS** on ajoute également le **userID** ainsi que le **bic** de l'institution dans laquelle le portefeuille a été créé dans la table **CLIENT_VS_INST** afin que l'institution qui n'a pas accès à la table **WALLETS** puisse tout de même savoir quels clients sont clients de son institution et n'ait accès qu'à ceux-ci dans la table.

Cette table possède une Primary Key qui est le **walletID** et qui attribue à chaque portefeuille d'un client un numéro de suite qui unique. Ce numéro permettra entre autres dans l'application client de récupérer tous les produits d'un portefeuille. Le **userID** référence la table **CLIENT** afin d'avoir ses informations dans son portefeuille.

La table **ACCOUNTS** contient les produits financiers de type Comptes bancaires. Chaque compte est identifié de manière unique grâce à son iban. Cette table référence la table **WALLETS** grâce au **walletID** qui permet de n'avoir que les produits d'un portefeuille.

La table **CO OWNERS** est une table contenant la liste de tous les co-titulaires. Dans celle-ci on retrouve l'**ibanOwner** qui référence vers la table **ACCOUNTS** et permet au possesseur du compte de savoir qui est co-titulaire et de le gérer. Cette table possède également le **walletID_coOwner** qui référence vers la table **WALLETS** et qui permet au co-titulaire de récupérer également les comptes qui ne sont pas en sa possession. Enfin il y a aussi le **userID_coOwner** qui permet aux institutions de voir les co-titulaires des comptes.

La table **CASH_BALANCES** contient les différents soldes d'un compte bancaire. En effet un compte dans l'application de base n'a qu'un solde EURO mais pour des soucis de simplicité de création de la base de données la possibilités d'avoir des soldes dans d'autres devises a déjà été ajoutée.

La clé primaire de cette table est le balanceID qui est un numéro de suite unique permettant d'identifier chaque solde d'une devise d'un compte. La currency pour une balanceID est unique dans cette table car un compte ne peut pas posséder plusieurs soldes dans la même devise. La table référence la table **ACCOUNTS** à l'aide de l'iban du compte afin d'obtenir tous les soldes d'un compte.

La table **TRANSACTIONS** est une liste de transactions à effectuer. Une transaction est composée de 2 flux (l'ascendant et le descendant) Mais dans ce cas, un élément de la table ne reprend qu'un seul des 2 flux car les 2 flux ne sont pas liés au même compte bancaire. C'est pourquoi il ya un colonne trxID qui est un numéro de suite de transaction. Ce numéro de suite est attribué aux 2 flux (streamID) d'une transaction et permet de vérifier que les 2 flux ont bien été exécutés avant d'appliquer les changements aux soldes.

Cette table référence la table **ACCOUNTS** à l'aide de l'iban d'un flux car c'est à partir d'un compte que l'on effectue une transaction. Ce même iban référence également la table **CASH_MVT**.

La table **CASH_MVT** est la table d'historique des transactions. Dans cette table ne se trouve que les mouvements qui ont été réalisés avec succès. Cette table est liée à la table transaction par l'iban afin d'ajouter un élément dès qu'une transaction est complétée.

C'est cette table qui se charge de mettre à jour les les soldes. Si un élément est ajouté à la table c'est que la transaction a bien eu lieu. Donc, grâce au balanceID qui référence la table **CASH_BALANCES** et au ammount on peut effectuer la mise à jour du solde du compte.

2.3.3 API

L'API est de type REST. Elle a été divisée en 3 endpoints (/user ; /wallet ; /financial_product) correspondant chacun à un type de donnée pouvant être édité.

Du à un manque de compréhension, nous n'avons pas utilisé les erreurs de la série 500 (problème serveur) dans le diagramme mais nous avons bien utilisé les erreurs de la série 400 correspondant à un problème dans la demande à l'API. Nous utilisons les erreurs 401 - Unauthorized , 403 - Forbidden , 404 - Not found , 409 - Conflict.

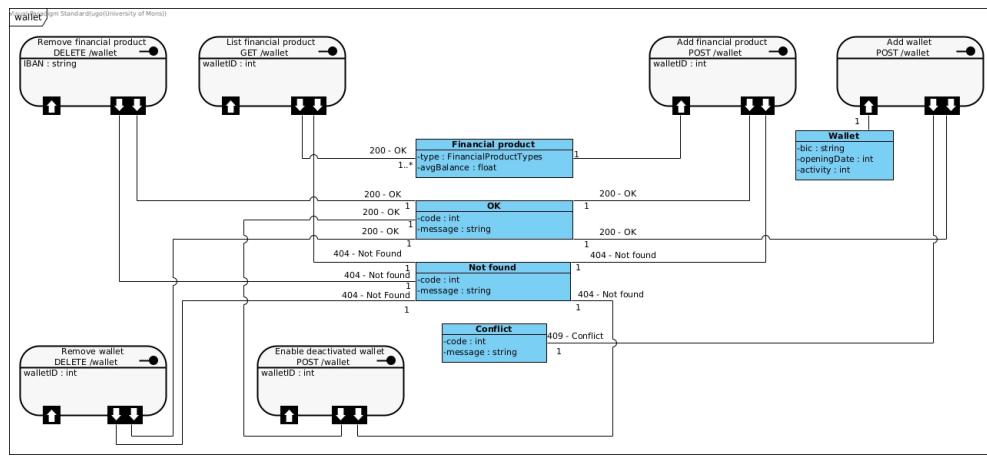


FIGURE 18 – API de l'endpoint /wallet

wallet permet d'interagir avec les wallet et une partie de leur contenu (lister le contenu ou y ajouter un produit financier).

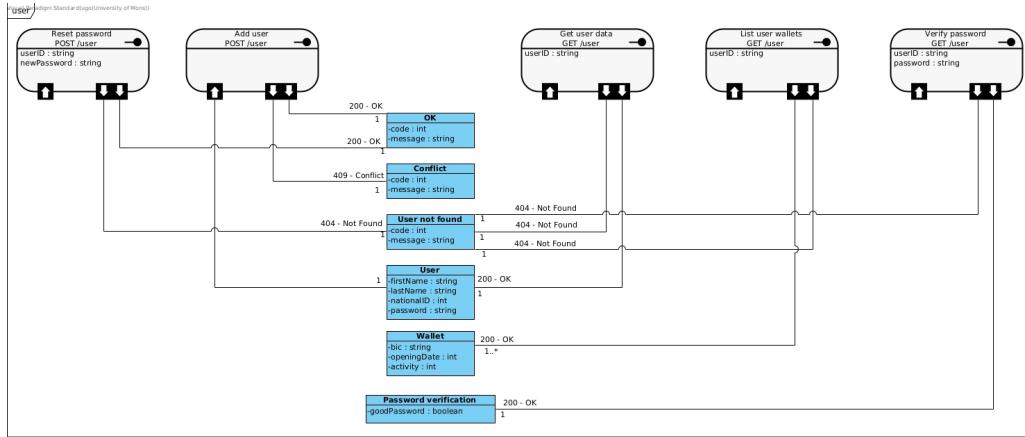


FIGURE 19 – API de l'endpoint /user

user permet d'intéragir avec les données d'un utilisateur, de lister ses wallets et de l'authentifier grâce à Verify password. La façon dont nous vérifions le mot de passe est à corriger car elle n'est pas sécurisée, le mot de passe est envoyé directement à l'API dans l'URI ce qui est vulnérable à une attaque du type "man in the middle" si une personne arrive à intercepter les requêtes. Même problème pour Reset password.

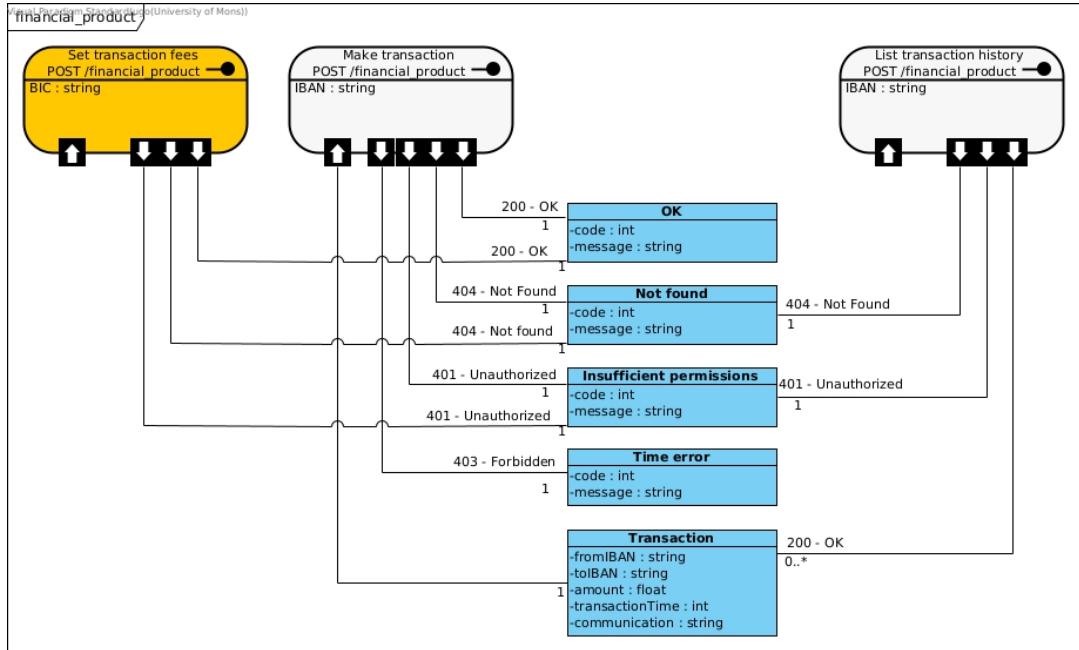


FIGURE 20 – API de l'endpoint `/financial_product`

`financial_product` intéragit directement avec le contenu des produits financiers permettant de faire des transactions ou de voir l'historique. Nous avons essayé d'intégrer un certain degré de sécurité en utilisant un système de permission, il n'est pas visible sur le schéma mais on peut tout de même voir que l'API est capable de renvoyer le code d'erreur 401 - Unauthorized.

Nous pouvons aussi observer que Make transaction peut renvoyer une erreur 403 - Forbidden, dans notre cas elle est exclusivement utilisée pour signaler que le temps demandé pour effectuer la transaction n'est pas bon. Il doit être soit à 0 pour dire que la transaction doit être faite immédiatement ou sur une valeur dans le futur pour indiquer le temps auquel la transaction doit être faite. (nous utilisons UNIX time)

2.4 Interface Graphique

2.4.1 Application 1

Cette section décrit les différents interfaces auxquels les utilisateurs de l'application auront accès pour gérer leurs comptes, portefeuilles et produits financiers.

1. LoginScreen

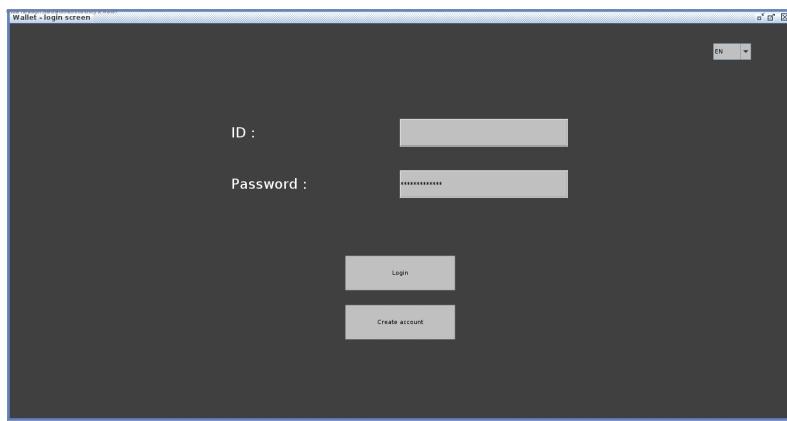


FIGURE 21 – Login Scene

L'écran de connexion est la première interface affichée lors du démarrage de l'application.

Cette interface permet à l'utilisateur de se connecter via ses identifiants ou d'accéder à l'écran de création de compte.

La langue d'affichage peut également être changée directement depuis le coin supérieur droit.

2. RegisterScreen

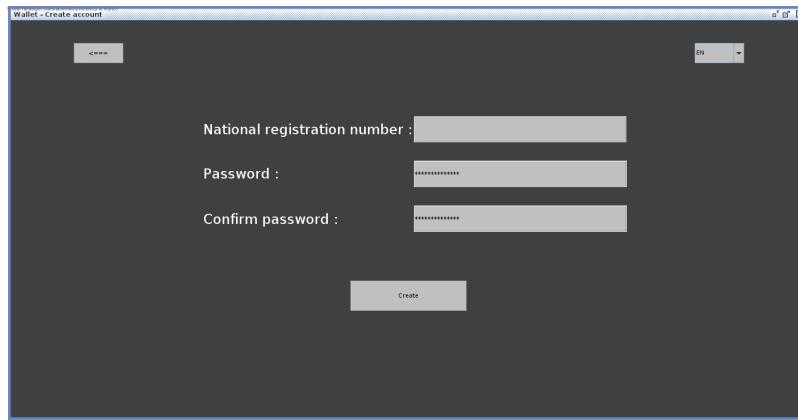


FIGURE 22 – Login Scene

Un utilisateur peut s'enregister via cet écran en entrant les données demandées.

Le numéro de registre national étant unique, le client ne sera pas en mesure de se créer plusieurs compte sur l'application.

Une fois le compte créé, l'utilisateur sera amené à un écran de connexion afin de procéder à l'authentification.

3. MainMenuScreen



FIGURE 23 – Login Scene

Une fois qu'un utilisateur s'est authentifié via le LoginScreen, le menu principal lui est affiché. Il permet à l'utilisateur d'accéder à toutes ses données via le menu à gauche de l'écran. Il contient un bouton "My wallets" qui mène l'utilisateur vers l'écran de gestion de ses portefeuilles financiers.

Un bouton "My account" permet à l'utilisateur d'avoir un écran d'aperçu de ses données. Il pourra aussi changer de mot de passe via cet écran.

Le bouton "Parameters" mène à l'écran des paramètres de l'app.

Le bouton "Deconnection" permet de se déconnecter afin de changer d'utilisateur (il mène à l'écran de connexion).

Et le bouton "Quit" déconnecte l'utilisateur et ferme l'application.

4. WalletsListScreen

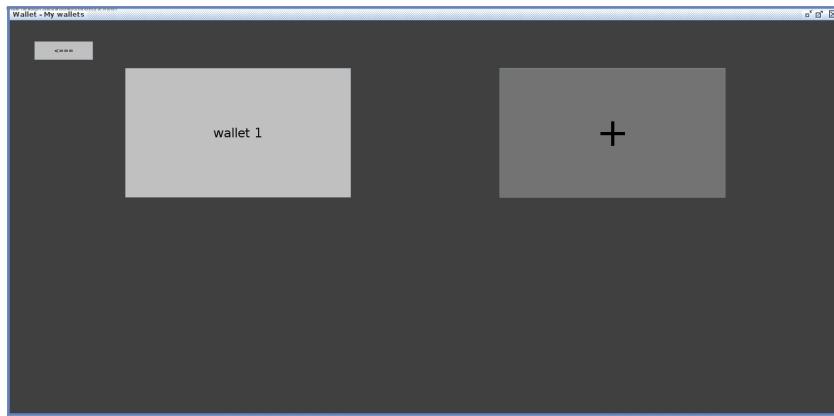


FIGURE 24 – Wallet List Scene

Cet écran affiche tous les portefeuilles de l'utilisateur ainsi qu'un bouton permettant d'en ajouter de nouveaux.

Les boutons de chaque portefeuille mènent l'utilisateur aux écrans affichant les détails de ceux-ci tandis que le bouton d'ajout de portefeuille envoie sur l'écran de création de portefeuille.

L'utilisateur ne pourra créer qu'un portefeuille par institution dont il est client.

Un bouton de retour en arrière est aussi présent sur l'écran des portefeuilles afin de permettre à l'utilisateur de retourner au menu principal.

5. AddWalletScreen

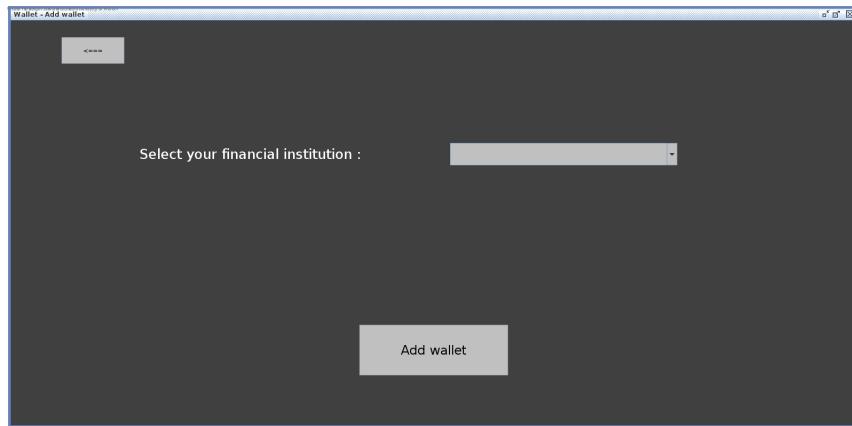


FIGURE 25 – Scene to add a wallet

AddWalletScreen propose à l'utilisateur de choisir une institution financière pour laquelle il veut avoir un portefeuille financier visible dans le menu des portefeuilles (WalletsListScreen). Si il est client de cette institution, la demande sera validée et le portefeuille sera accessible.
Un bouton retour de en arrière se trouve dans le coin supérieur gauche de l'écran. Ce bouton renvoie vers la liste des portefeuilles de l'utilisateur.

6. AddProductScreen

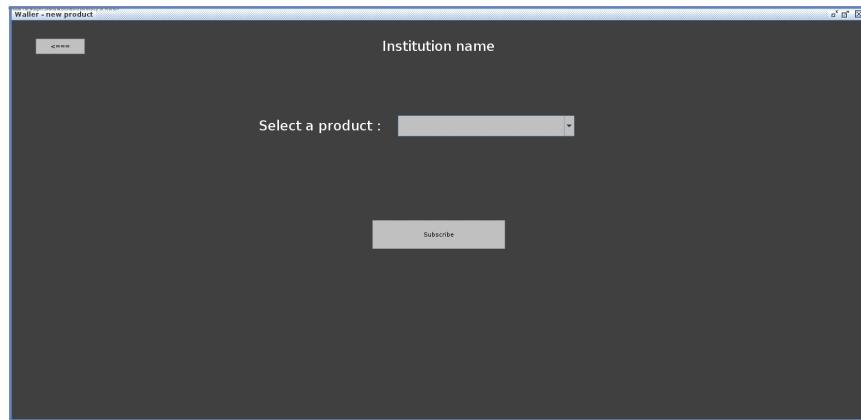


FIGURE 26 – Scene to add a product

Afin d'ajouter un produit financier au portefeuille sélectionné, l'utilisateur doit choisir le type de produit financier qu'il veut créer via le menu déroulant de cet écran.

Il pourra ensuite cliquer sur "add product" pour conclure l'opération. Un bouton retour arrière lui permet d'annuler la création du produit et revenir au menu du portefeuille sélectionné.

7. ProductsListScreen

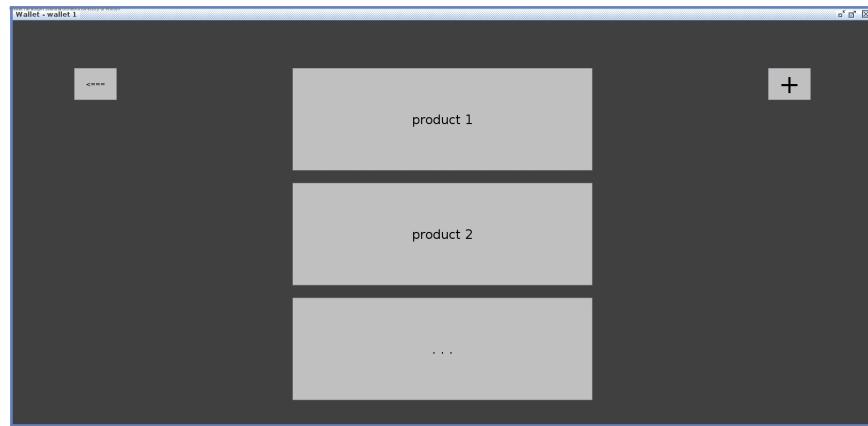


FIGURE 27 – List of the products

Chaque bouton permet à l'utilisateur de consulter les détails d'un de ses produits financiers.

Il peut faire une demande d'ajout de nouveau produit financier. Une fois que celle-ci sera validée par l'institution, le produit créé apparaîtra dans la liste du menu précédent.

Un bouton de retour en arrière permet de revenir au menu précédent manuellement et donc d'annuler la création du produit financier.

8. ProductMenuScreen

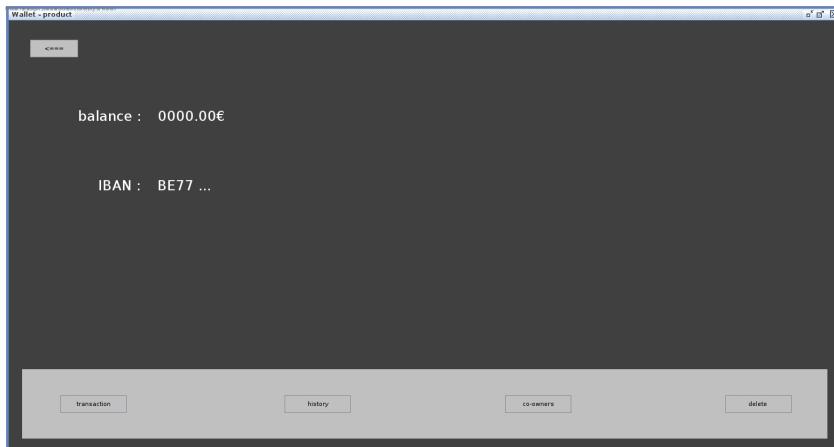


FIGURE 28 – Product menu

Cet écran est affiché à l'utilisateur lorsqu'il clique sur un de ses produits financiers.

Les informations générales du produit financier (telles que le solde du compte et le numéro de compte) y sont disposées et les boutons du bas de l'écran permettent à l'utilisateur de procéder à différentes opérations sur ce produit financier.

Effectuer une transaction se fait via le bouton "transaction", consulter l'historique des transactions se fait via le bouton "history", la gestion des co-titulaires se fait via le bouton "co-owners" et enfin la suppression du produit se fait via le bouton "delete".

Le bouton "delete" demandera à l'utilisateur de confirmer son action avant de procéder à la désactivation du produit.

Un bouton de retour en arrière est également disponible afin de revenir à l'écran du portefeuille dans lequel se trouve le produit financier courant.

9. TransactionScreen

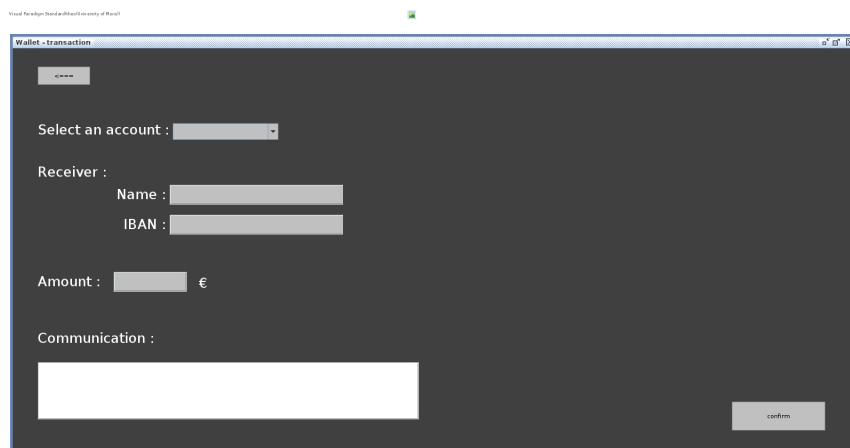


FIGURE 29 – Transaction scene

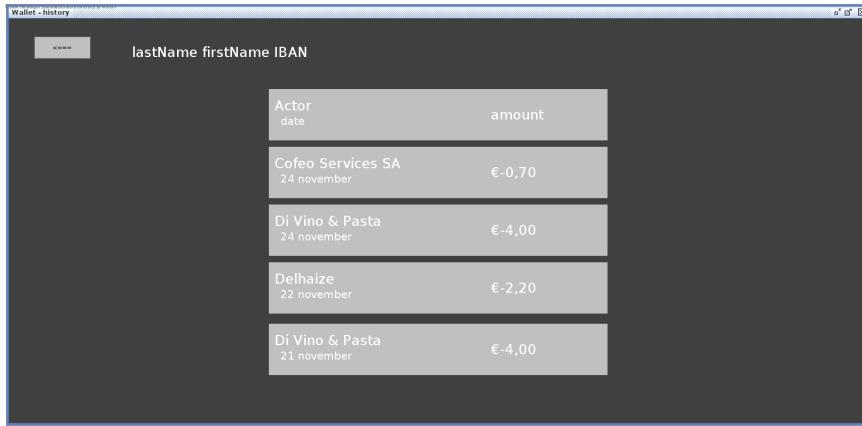
L'écran de transaction demande plusieurs informations à l'utilisateur afin d'effectuer une transaction d'un compte A à un compte B.

Si ce menu a été accéder via le menu d'un produit financier, le compte envoyeur ne peut pas être modifié par l'utilisateur. Sinon, il doit sélectionner un compte parmis tous ceux qu'il possède.

Le nom et l'IBAN du receveur doit ensuite être entré dans leurs champs respectifs. Finalement, le montant de la transaction doit être précisé et optionnellement une communication peut être jointe à la transaction.

Une fois toutes les données entrées, le bouton "confirm" va afficher un résumé de la transaction, l'utilisateur pourra alors décider de confirmer ou annuler la transaction. Le bouton retour arrière renvoie vers le menu du produit financier et annule la transaction courante.

10. ProductHistoryScreen



Actor date	amount
Cofeo Services SA 24 november	€-0,70
Di Vino & Pasta 24 november	€-4,00
Delhaize 22 november	€-2,20
Di Vino & Pasta 21 november	€-4,00

FIGURE 30 – History of the product

Cet écran affiche toutes les transactions effectuées depuis et vers le produit courrant.

Pour chaque transaction, l'envoyeur ou destinataire est affiché ainsi que la date de l'opération et le montant.

Le bouton retour arrière mènera l'utilisateur vers le menu du produit.

11. OwnersScreen

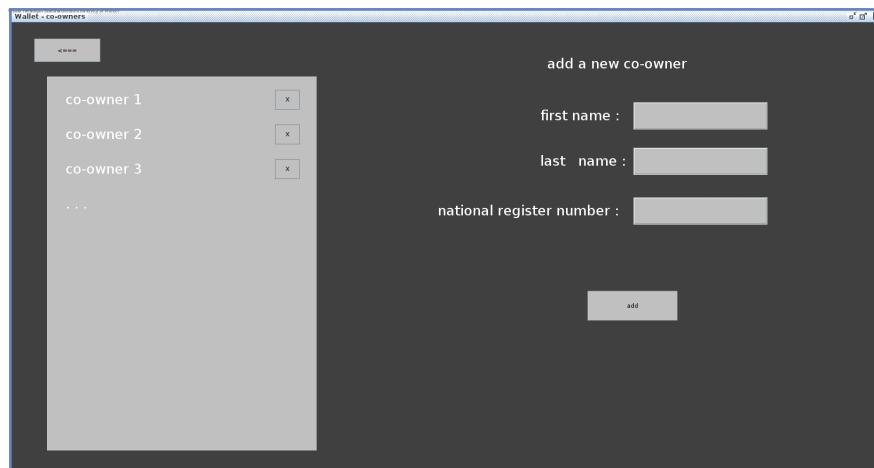


FIGURE 31 – Owners scene

L'écran de gestion des co-titulaires affiche une liste de toutes les personnes partageant le compte.

L'utilisateur peut supprimer ou ajouter des co-titulaire via ce menu.

Le nom, prénom et numéro de registre national sont nécessaires afin d'ajouter un nouveau co-titulaire.

Le bouton de retour arrière renvoie l'utilisateur vers le menu du produit.

2.4.2 Application 2

2.4.3 Application clients

Cette section décrit les différents interfaces auxquels les employés de l'application auront accès pour gérer leurs clients et leurs produits financier.

1. LoginScene

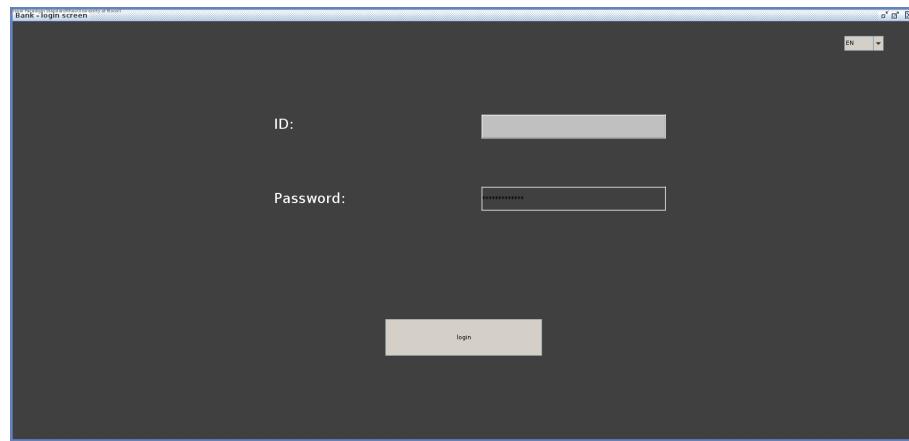


FIGURE 32 – Login Scene

L'écran de connexion est la première interface affichée lors du démarrage de l'application.

Cette interface permet à l'employé de l'institution de se connecter via ses identifiants .

La langue d'affichage peut également être changée directement depuis le coin supérieur droit.

2. MainMenuScreen

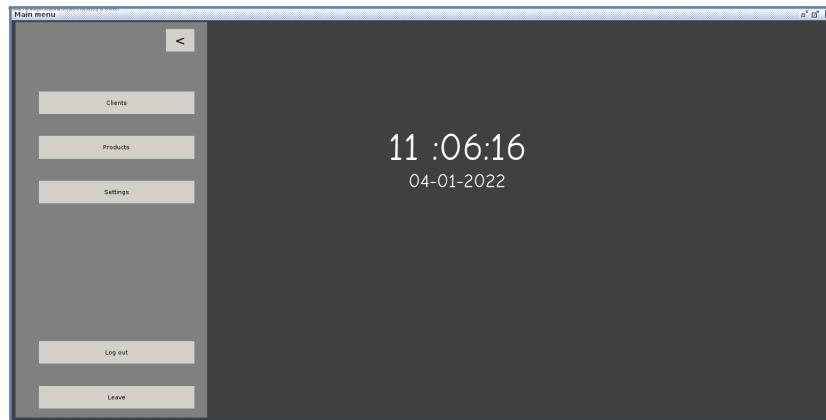


FIGURE 33 – Main Menu

Une fois qu'un employé s'est authentifié via le LoginScene, le menu principal lui est affiché. Il permet à l'employé d'accéder au données de l'institution via le menu à gauche de l'écran. Il contient un bouton *Clients* qui mène l'employé vers l'écran de listant tout les clients de l'institution. Un bouton *Products* permet à l'employé d'avoir un accès a la liste de tout les produits possédé par des clients. Il pourra y ajouter ou supprimer des produits via cet écran.

Le bouton *Settings* mène à l'écran des paramètres de l'app.

Le bouton *Log out* permet de se déconnecter (il mène à l'écran de connexion). Et le bouton *Quit* déconnecte l'employé et ferme l'application.

3. ClientMenuScene

Name	#1	#2	Registration date	# of products
Client 1	#1		01-01-21	7
Client 2	#2		02-02-21	3
Client 3	#3		03-03-21	0
...				

FIGURE 34 – List of the clients

L'employé se voit afficher une liste de tout les clients et peut les gerer graces au différents boutons.

Le bouton *Add Client* redirige vers le menu d'ajout de clients *addClientMenu*

Le bouton *Delete Client* redirige l'employé vers le menu de suppression de client *DeleteClientMenu*

Le bouton *back* qui permet de retourner vers *MainMenuScreen*

Un clique droit sur un client ouvre le menu supplémentaire *rightClickOnClient*

L'employé peut aussi trier les clients en fonctions des différentes caractéristiques listé a l'écran.

4. addClientMenu

The screenshot displays a user interface for adding a new client. At the top, a title bar reads "Enter the new client's informations". Below the title, there are three input fields arranged vertically. Each field has a label on its left and a corresponding text input box on its right. The first field is labeled "First name:", the second "Last name:", and the third "ID:". At the bottom of the form is a single button labeled "Add".

FIGURE 35 – Add client menu

L'employé peut ajouter un nouveau client en entrant le nom (Last name), le prénom (First name) et le numéro de registre national (ID). Le bouton *Add* permet d'ajouter le client à la liste des clients.

5. delClientMenu



FIGURE 36 – Add client menu

L'employé peut supprimer un client en entrant l'ID (nom_prenom_nrRegistreNational) de celui-ci.

Le bouton *Delete* permet de supprimer le client de la liste des clients.

6. rightClickOnClient



FIGURE 37 – Right click on a client menu

L’employé peut gerer les clients et leurs produits rapidement grâce à ce menu.

Le bouton *delete client* va supprimer le client sur lequel l’employé a effectué le clique droit.

Le bouton *products detail* va afficher les produits du client sur lequel l’employé a effectué le clique droit.

Le bouton *Permit transactions* permettra de rediriger l’employé vers le menu d’autoriser des virements du client sur lequel le clique droit a été effectué.

Le bouton *add product* permettra de rediriger l’employé vers le menu d’ajout de produit financier pour le client sur lequel le clique droit a été effectué.

Le bouton *delete product* permettra de rediriger l’employé vers le menu de suppression de produit financier pour le client sur lequel le clique droit a été effectué.

7. Allow Transaction



FIGURE 38 – Allow transaction menu

Ce menu permet à l'employé d'autoriser ou non les transactions du client.

8. ProductMenu Scene

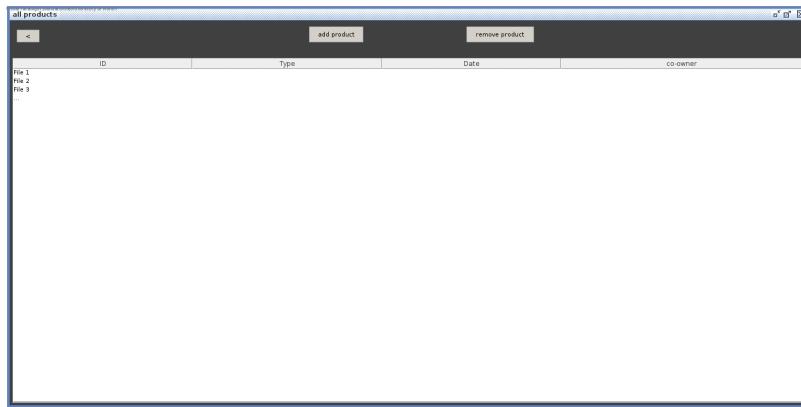


FIGURE 39 – List of all subscribed to products

Cette scene donne la l'employé la poossibilité de gerer les produits des clients.

Un clique sur un produit montrera à l'employé "Clique on a product" avec des informations supplémentaire du produit.

Le bouton *add product* permettra de rediriger l'employé vers le menu d'ajout de produit financier pour le client sur le quel le clique droit a été effectué.

Le bouton *delete product* permettra de rediriger l'employé vers le menu de suppression de produit financier pour le client sur le quel le clique droit a été effectué.

Si l'employé effectue un clique droit sur un produit il accède a un autre menu.

9. add product Scene

Ce menu permet à l'employé d'ajouter un nouveau produit financier à

The screenshot shows a user interface for adding a new financial product. The title bar reads "Add Product [Add a new entry of Product]". The form contains four input fields: "client ID :" with a text input field, "type of product :" with a dropdown menu, "initial balance :" with a text input field, and "co-owner :" with a text input field. Below these fields are two checkboxes: "adult" and "minor". At the bottom right is a "Confirm" button.

FIGURE 40 – Add product menu

un client en y entrant l'ID du client, le sold initial, un co-titulaire éventuelle, si le client est majeur ou pas et de sélectionner le type de produit à ajouter.

Le bouton *Confirm* finalise la procédure et ajoute le produit financier au client.

10. delete product Scene

Ce menu permet à l'employé de supprimer un produit financier d'un



FIGURE 41 – product menu

client en y entrant l'ID du client et l'IBAN du produit à supprimer.
Le bouton *Confirm* finalise la procédure et supprime le produit financier
du client.

11. Client product Scene

Cette scene affiche tout les produits financier au quel un client est sous-

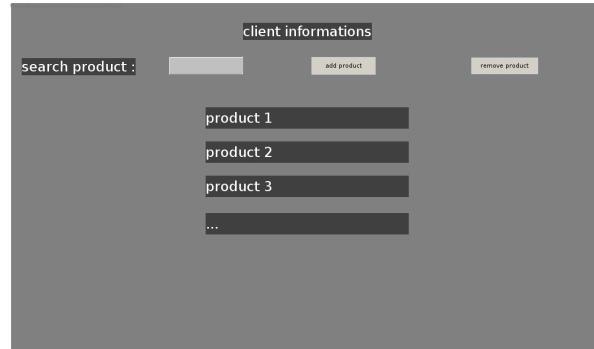


FIGURE 42 – Products of a client menu

crit et permet à l'employé de les trier en fonction de leurs nom.

Le bouton *add product* redirige l'employé vers *add product scene*.

Le bouton *remove product* redirige l'employé vers *delete product scene*.

12. Clique on a product

Ce menu affiche des informations supplémentaires sur un produit financier d'un client.



FIGURE 43 – Details of a Product

cier d'un client.

3 Extensions

3.1 Extension 1, Gestion des cartes - Godin Théo :

3.1.1 Application 1

3.1.2 Use cases de l'application 1

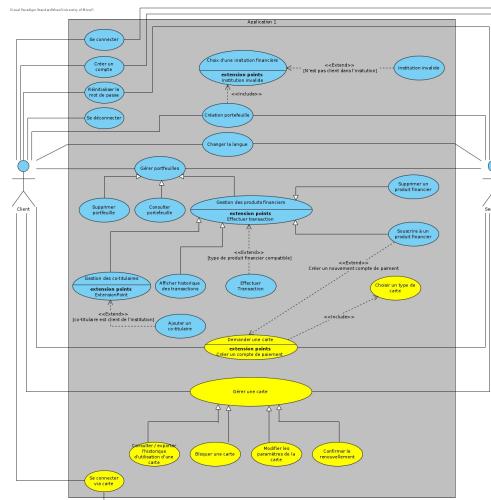


FIGURE 44 – Use cases de l'application 1

Les principaux use cases ajoutés afin de supporter les cartes de paiement sont ajouter une carte, gérer une carte et se connecter avec une carte.

3.1.3 Interaction overview de l'application 1

Les use cases de gestion de cartes ont été ajouté au diagramme d'interaction. Ils sont accessibles via le menu des cartes disponibles dans le menu principal.

3.1.4 Diagramme de classes de l'application 1

Dans le package APP, des classes représentant les différents types de cartes ont été ajoutées. Les cartes de débits et de crédits partagent la même classe mère représentant une carte de paiement.

Plusieurs scènes ont été ajoutées à l'interface graphique afin de proposer à l'utilisateur des menus de gestion de ses cartes de paiement.

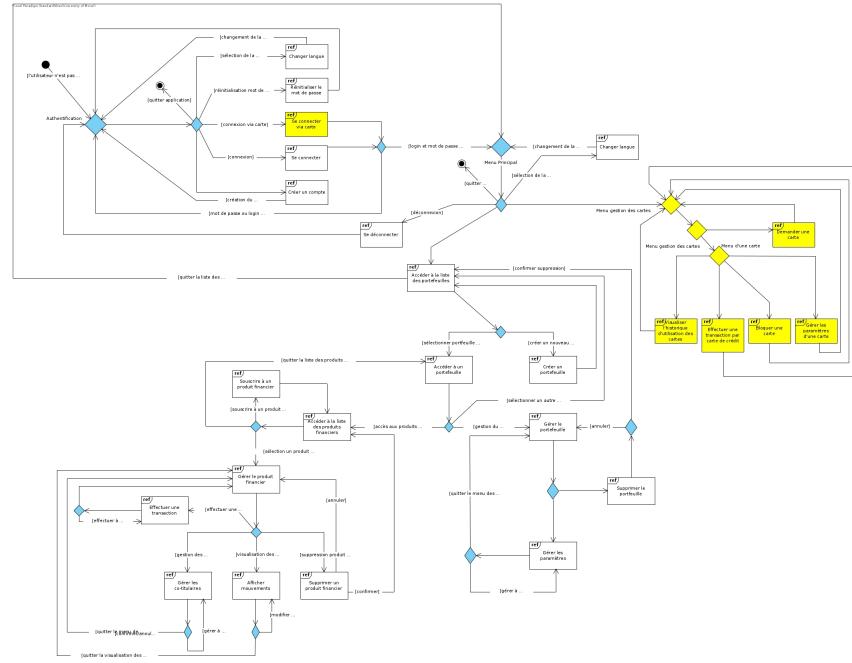


FIGURE 45 – Interaction overview diagram de l’application 1

3.1.5 Diagrammes de séquences de l’application 1

1. Se connecter via une carte

Afin de se connecter avec une carte, l’utilisateur doit insérer sa carte dans le lecteur de carte. Il clique ensuite sur login et entre le code affiché par l’application. Ce code est généré via une méthode de l’api. Il entre ensuite son code pin. Si l’autentification est validée, le lecteur affiche une code qui permettra à l’utilisateur de se connecter à l’application.

3.1.6 Interface graphique de l’application 1

1. addCardScreen

2. cardLoginScreen

Ce menu permet à l’utilisateur de se connecter avec une carte.
La partie droite de l’écran représente un lecteur de carte afin de simuler

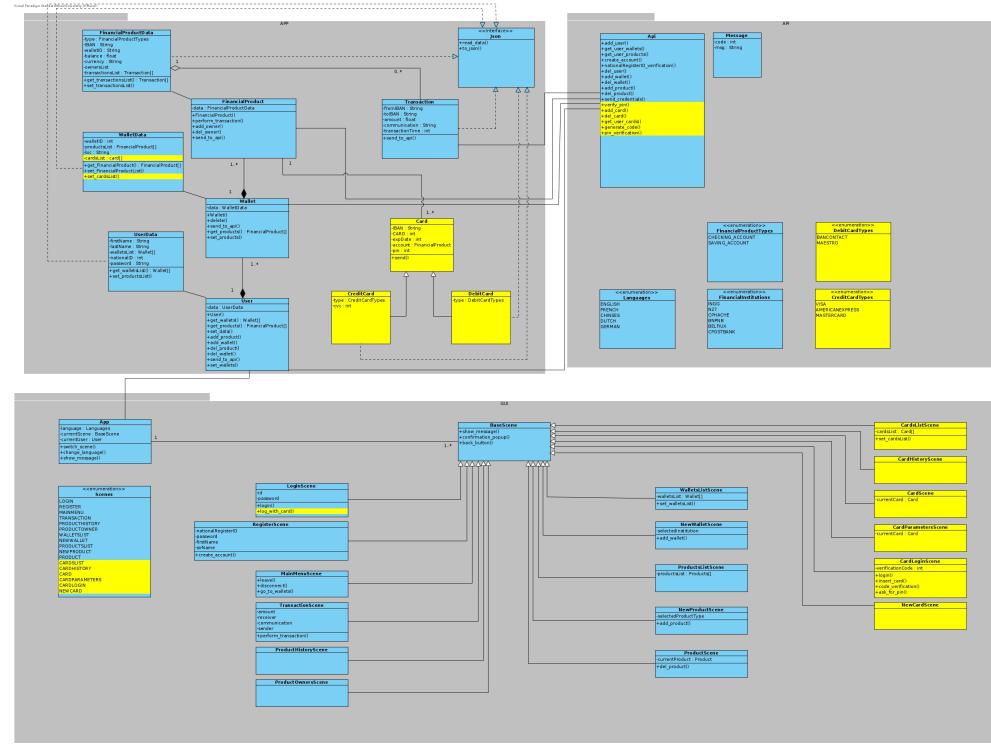


FIGURE 46 – Class diagram de l'application 1

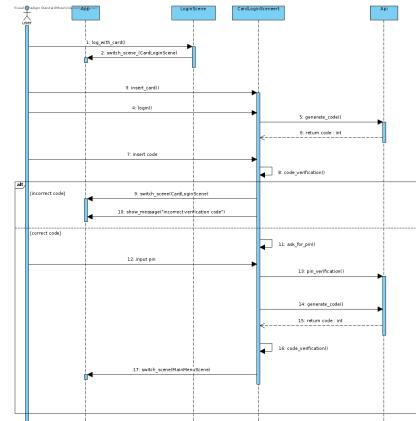


FIGURE 47 – Se connecter via une carte

l'insertion d'une carte.

3. cardScreen

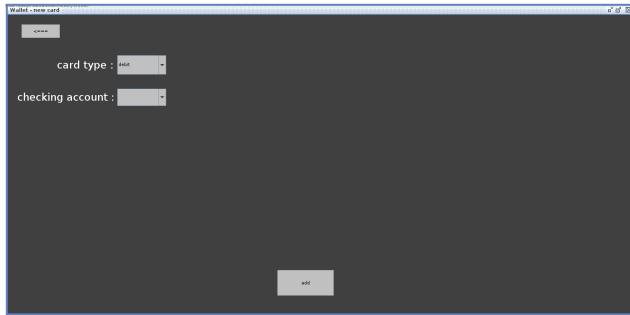


FIGURE 48 – Écran d'ajout de carte



FIGURE 49 – Écran de connexion via carte

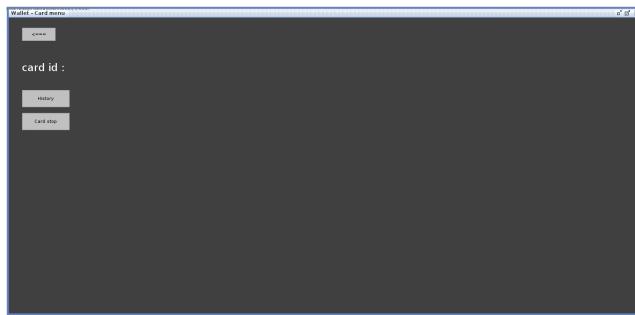


FIGURE 50 – Écran de gestion d'une carte

4. cardListScreen

Cet écran liste toutes les cartes de l'utilisateur.

Il peut cliquer sur n'importe quelle carte afin d'accéder au menu de cette carte.

5. cardsParametersScreen

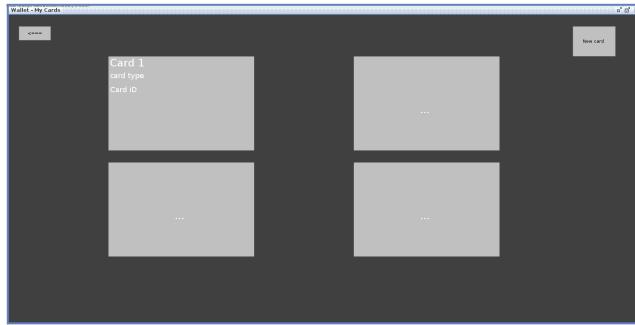


FIGURE 51 – Écran de la liste de cartes

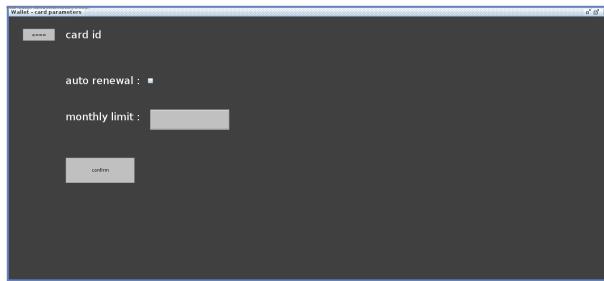


FIGURE 52 – Écran de paramètres d'une carte

Cet écran permet de modifier les paramètres d'une carte tel que le plafond mensuel et le renouvellement automatique.

6. loginScreen



FIGURE 53 – Écran connexion

Le bouton “login with card” permet à l’utilisateur d’accéder au menu de

connexion par carte.

7. mainMenuScreen

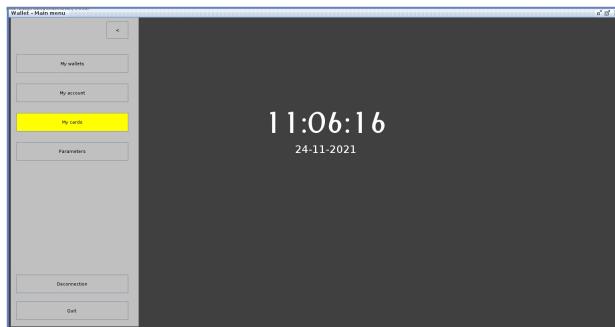


FIGURE 54 – Écran principal

Le bouton “My cards” permet à l’utilisateur d'accéder au menu de gestion des cartes.

3.1.7 Application 2

3.1.8 Use cases de l’application 2

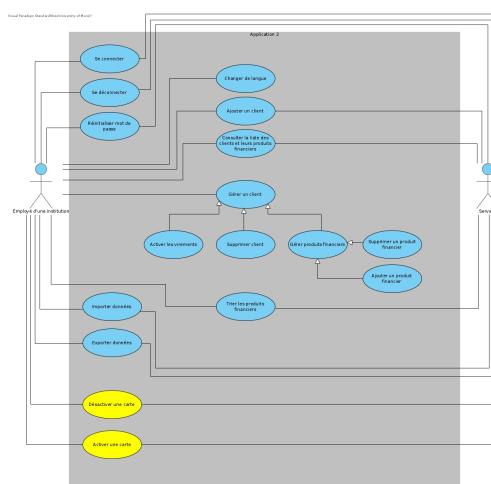


FIGURE 55 – Use cases de l’application 2

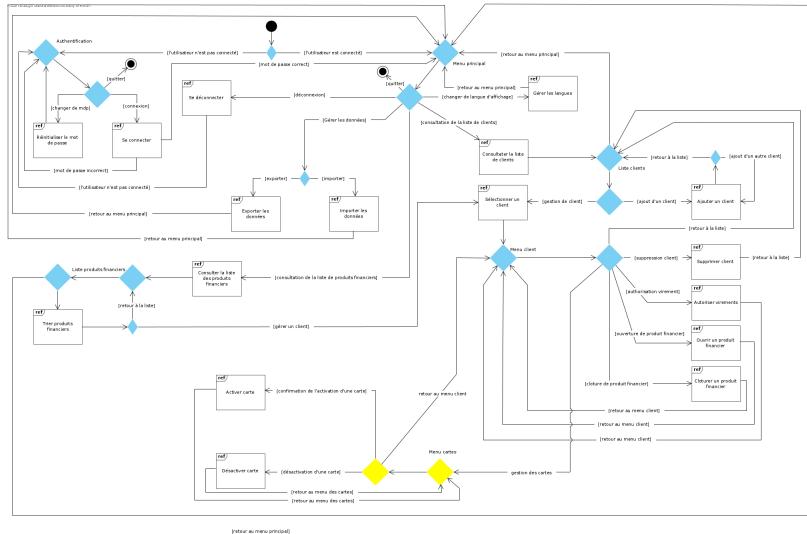


FIGURE 56 – Interaction overview de l’application 2

3.1.9 Interaction overview de l’application 2

3.1.10 Diagramme de classes de l’application 2

3.1.11 Interface graphique de l’application 2

1. CardMenuScreen

2. clientProductScreen

3.1.12 Diagramme d’entité relation

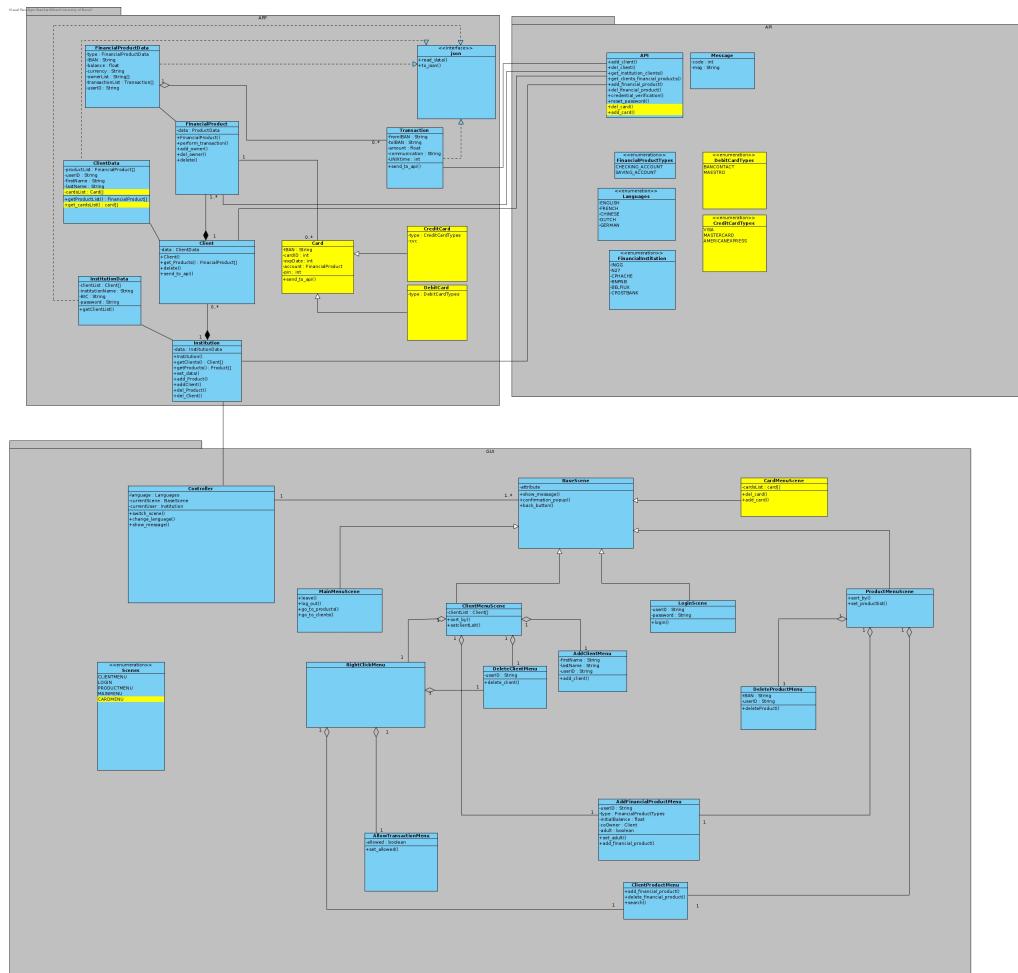


FIGURE 57 – Interaction overview de l'application 2

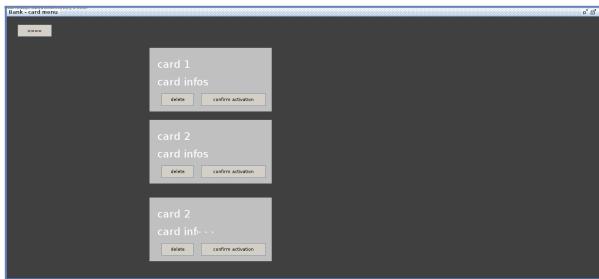
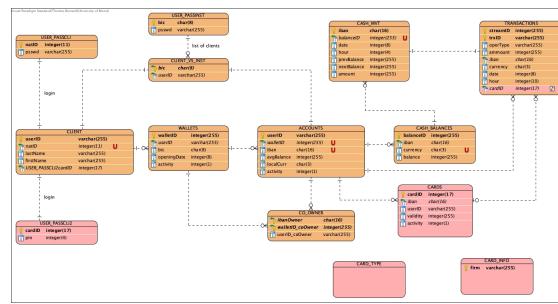


FIGURE 58 – Écran de gestion des cartes



FIGURE 59 – Écran des produits d'un client



3.2 Extension 2, Gestion des devises et virements internationaux - Projet Ugo :

3.2.1 Diagramme des cas d'utilisation de l'application 1

Ici il a simplement fallu ajouter 2 utilisations : Afficher l'historique des taux de conversion et calculer un taux de conversion. Nous verrons plus loin que tout est implémenté sur la même Scène.

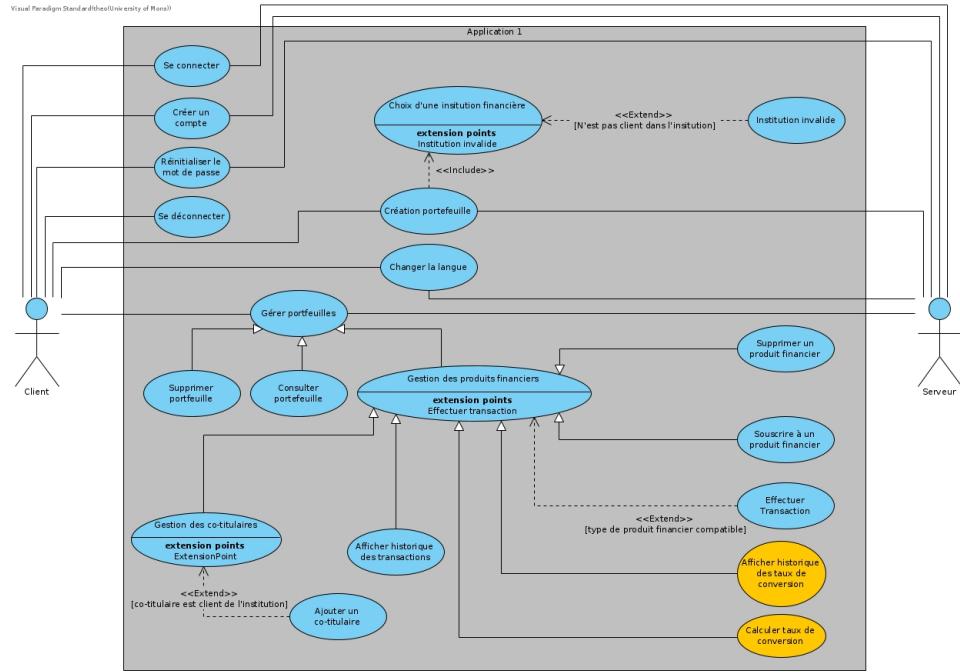
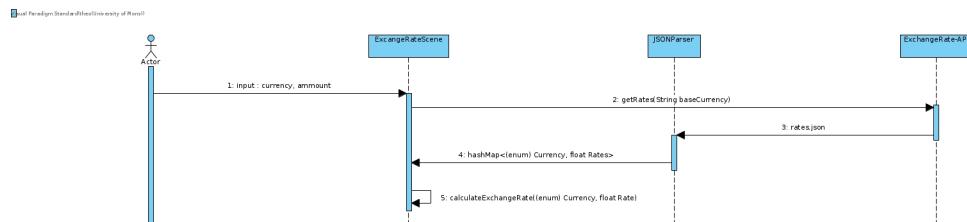


FIGURE 60 –

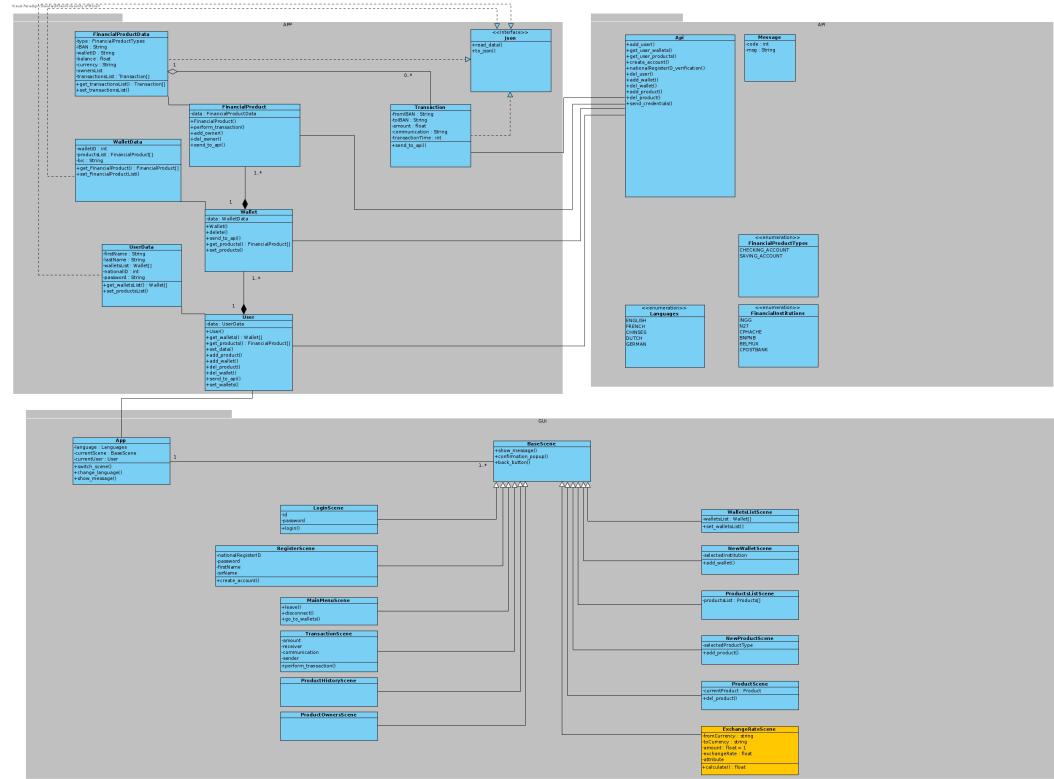
3.2.2 Diagramme de séquence

Un seul diagramme de séquence est nécessaire, celui ci décrit comment l'utilisateur interagit pour calculer un taux de change. On peut aussi noter l'utilisation d'une API externe (ExchangeRate-API) et d'un parser JSON afin de bien formater les données.



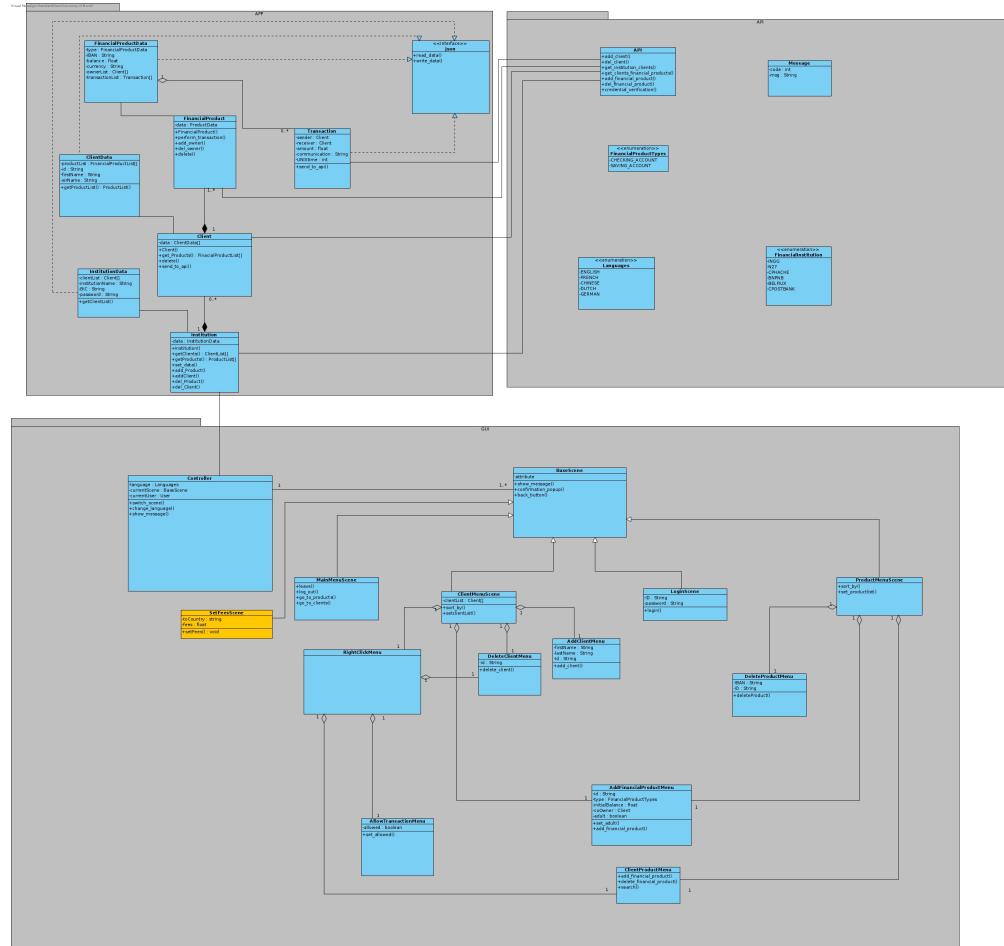
3.2.3 Classes de l'application 1

L'ajout d'une fenêtre dans le GUI de l'application 1 implique l'ajout d'une nouvelle classe contenant des variables (qui serviront à stocker les données relatives au taux de change) et une méthode qui servira à afficher le résultat.



3.2.4 Classes de l'application 2

L'ajout d'une fenêtre dans le GUI de l'application 2 implique l'ajout d'une nouvelle classe contenant des variables (qui serviront à stocker les données relatives aux frais de transaction) et une méthode qui servira à définir les frais (et les envoyer à l'API).

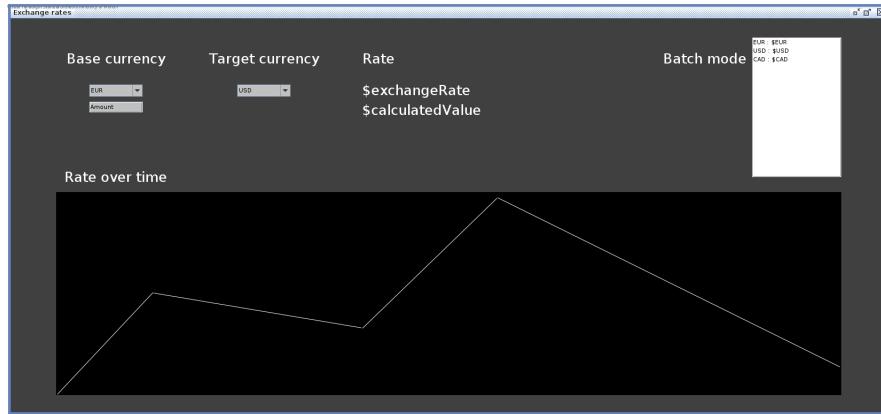


3.2.5 Interface graphique de l'application 1

Sur cette Scene, l'utilisateur peut introduire une monnaie et un montant (en dessous du texte Base currency), introduire une monnaie destination et voir le taux de change utilisé ainsi que la valeur finale.

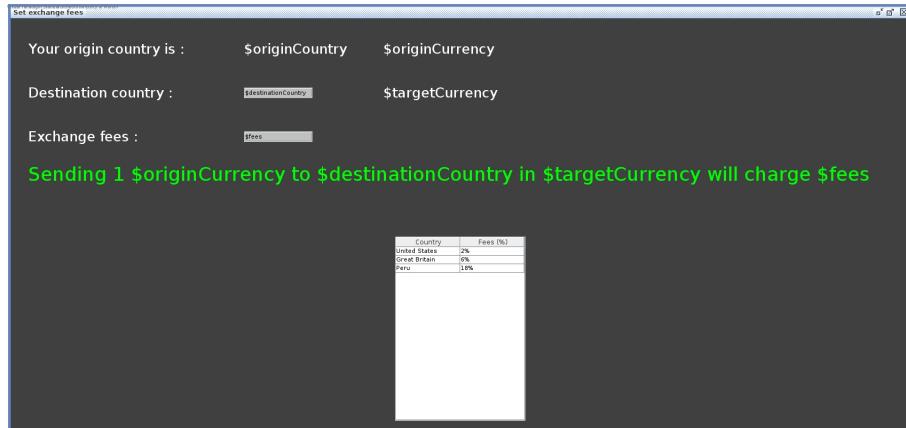
La partie batch mode est très pratique car elle affiche le taux de change de plusieurs monnaies afin d'avoir une vue globale rapide.

Le graphique en bas affiche le taux de change des deux monnaies choisies au dessus dans le temps.



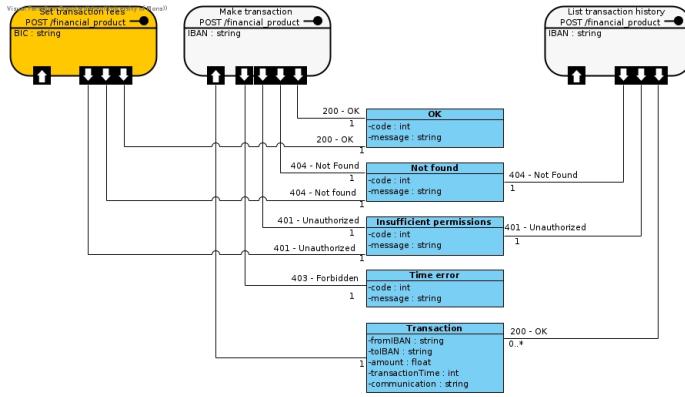
3.2.6 Interface graphique de l'application 2

Cette fenêtre est assez simple, l'institution entre un pays de destination ainsi que les frais. Les données sont inscrites dans la base de données et un message (en vert) confirme le changement.



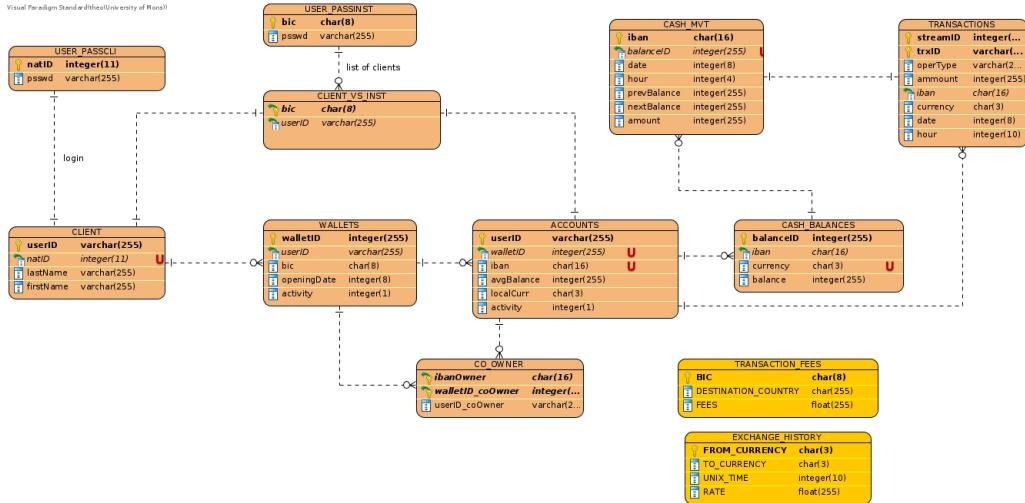
3.2.7 Diagramme de l'API (financial product)

Comme expliqué juste avant, une institution doit pouvoir enregistrer dans la base de données des frais de transaction. Il nous faut donc cette fonction dans l'API.

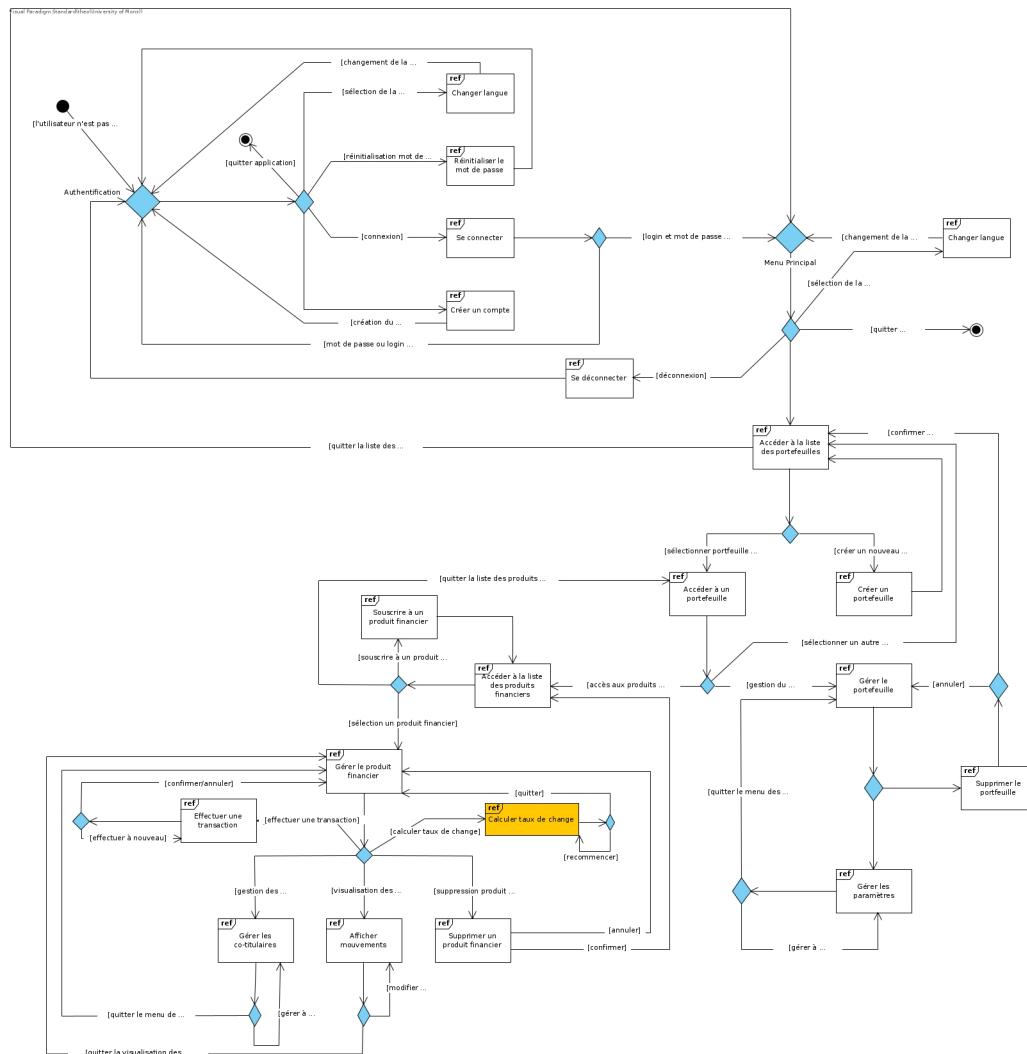


3.2.8 Base de données

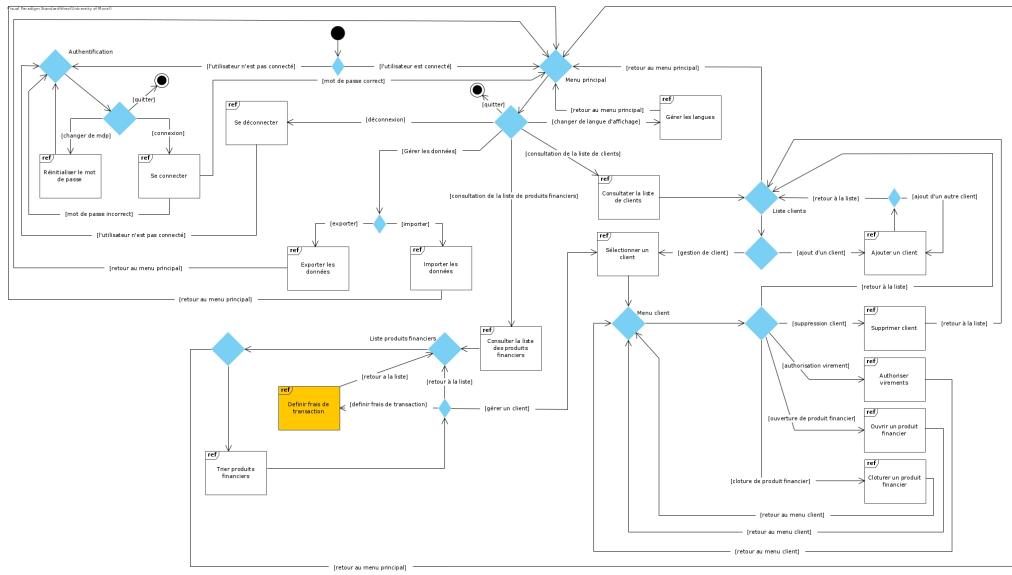
Une table permet de stocker les frais de transaction. Une autre pour garder l'historique des taux de change car ExchangeRate-API ne le permet pas.



3.2.9 Interaction Overview Diagram 1



3.2.10 Interaction Overview Diagram 2



3.3 Extension 5, Gestion des contrats d'assurance - Bernard Thomas :

3.3.1 Vue d'ensemble

Le but de cette extension est de rajouter la gestion de contrats d'assurances divers à la fois pour les clients mais également pour les institutions. L'extension se base tout de même sur la structure de l'application 1 car c'est dans celle-ci qu'elle la plus utilisée. En effet, dans l'application 2 elle est gérée comme les autres produits financiers. Il n'y a réellement que la réponse à une demande de devis qui diffère. Afin de rendre les diagrammes plus visibles j'ai changé les couleurs des éléments rajoutés. Rose pour le diagramme d'entité relation et jaune pour les autres diagrammes.

3.3.2 Application 1

3.3.3 Diagramme des cas d'utilisation

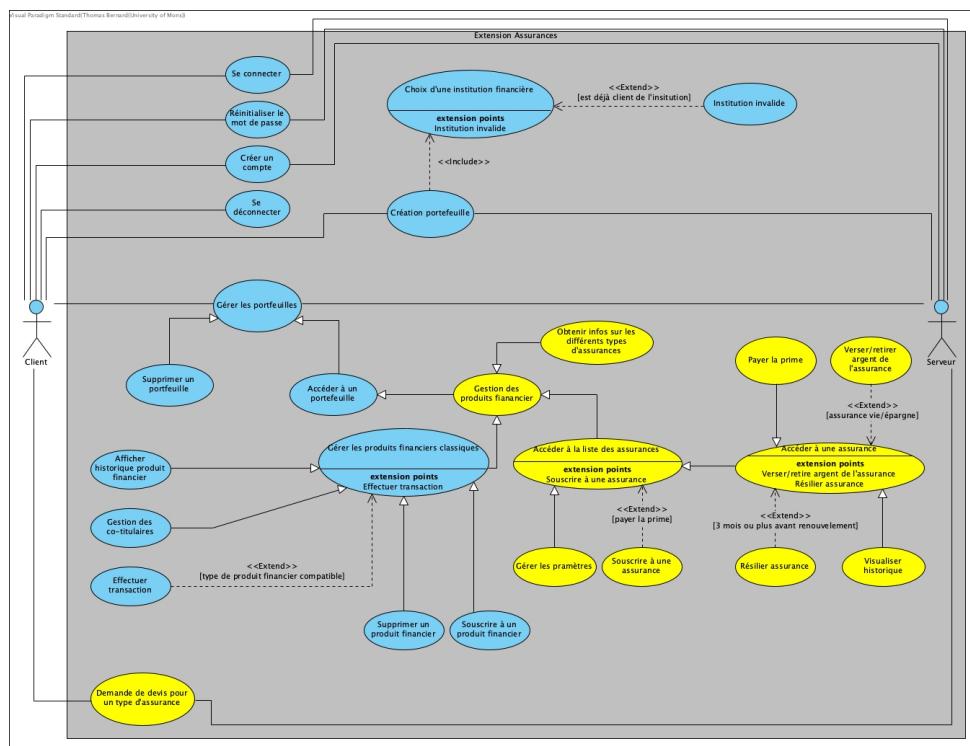


FIGURE 61 – Diagramme des cas d'utilisation de l'app 1 avec extension

J'ai rajouté un ensemble de divers cas d'utilisations qui sont propres aux assurances mais toutefois semblables aux cas d'utilisation relatifs aux produits

financiers classiques. Il n'y a aucune remarque particulière à faire concernant les cas d'utilisation car le modèle est calqué sur le diagramme de base.

3.3.4 Interaction Overview Diagram

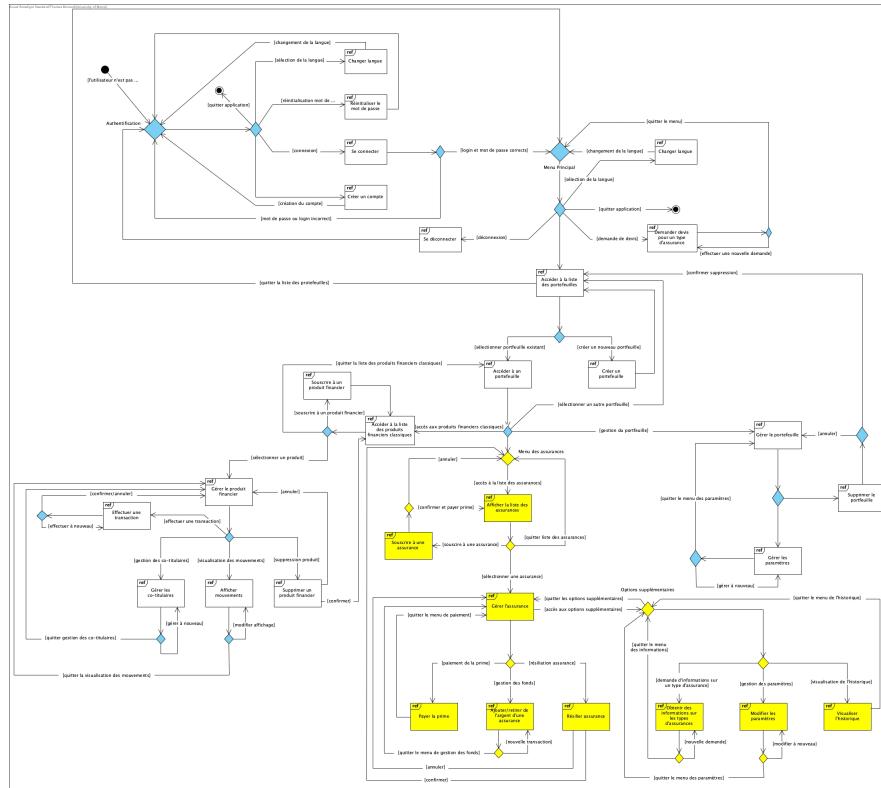


FIGURE 62 – Interaction Overview Diagram de l'app 1 avec extension

Encore une fois ce diagramme se base sur celui de l'application 1. Notons qu'il est maintenant possible de demander un devis depuis le menu principal. Il y a également une différenciation entre les produits financiers classiques et les assurances afin de les séparer en deux scènes différentes plus tard dans la GUI.

3.3.5 Diagrammes de classe

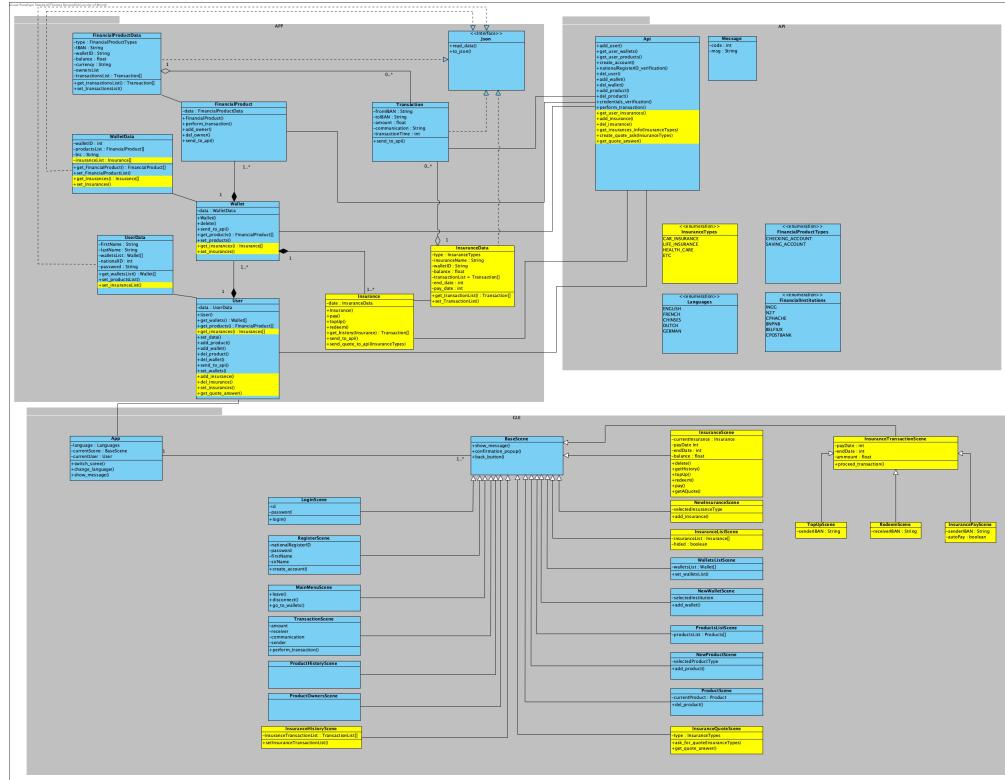


FIGURE 63 – Diagramme de classes de l'app 1 avec extension

Dans la partie logique de l'application (package APP) deux classes ont été rajoutées : **Insurance** et **InsuranceData**. La première contient le constructeur et un attribut *data* qui est une objet de type *InsuranceData*. La classe **Insurance** est liée à la classe *wallet* de même manière que la classe **FinancialProduct**. La classe **Insurance** possède une méthode *send_to_api()* et est connectée au package API. La classe **InsuranceData** implémente l'interface **Json** qui permet d'écrire et le dire des fichiers :json de données et d'ainsir mettre ses données à jour.

Les classes de bases dans lesquelles se trouvaient des listes de produits, des setter et des getter se sont vues ajouter des setter, des getter et des listes mais cette fois-ci pour les assurances.

Dans la partie serveur du diagramme de classe (package API) j'ai rajouté une enumération **InsurancesTypes** qui contient les différents types d'assurances

afin de pouvoir interpréter correctement les données envoyées à la classe **Api**. Dans la classe **Api** des méthodes ont été rajoutées afin de pouvoir ajouter, supprimer et obtenir les assurances d'un client. Il y a également une méthode permettant d'obtenir des informations sur un ou plusieurs types d'assurances dans le cadre de devis.

Dans la partie interface graphique (package GUI) il s'agit principalement de nouvelles scènes rajoutées à la manière des scènes de l'application de base. Il n'y a pas de point particulier à expliquer ici.

3.3.6 Diagrammes de séquence

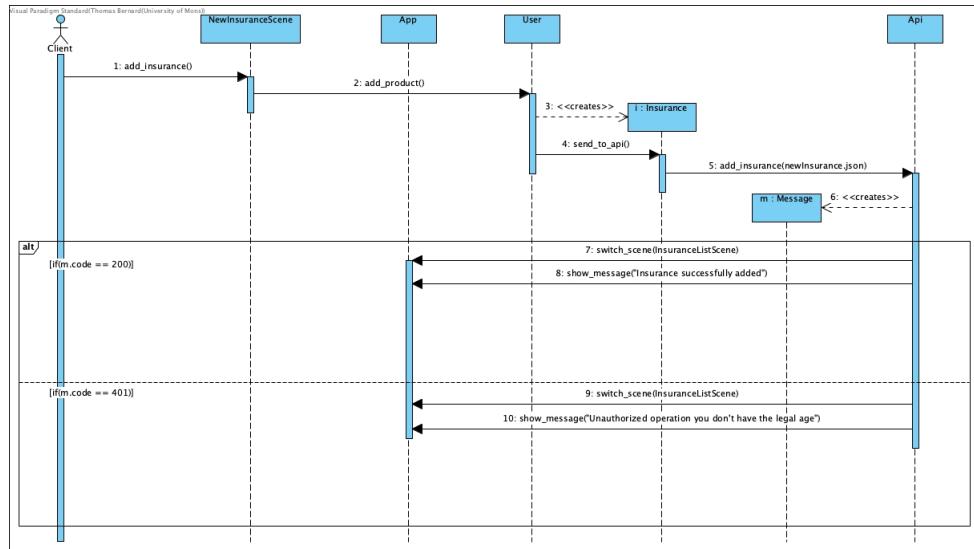


FIGURE 64 – Souscrire à une assurance

Souscrire à une assurance : Lorsque le client souhaite souscrire à une assurance l'application envoie à l'API une requête contenant les informations du demandeur ainsi que les informations de l'assurance demandée. L'API ajoute l'assurance à la base de données. Si l'ajout a bien eu lieu alors la scène change sur celle de la liste des assurances qui lors de sa création récupère les assurances du client. L'application affiche le message qui dit que l'assurance a bien été ajoutée. Si l'ajout n'a pas lieu le client est redirigé sur la scène de la liste des assurances et un message l'informe de l'erreur.

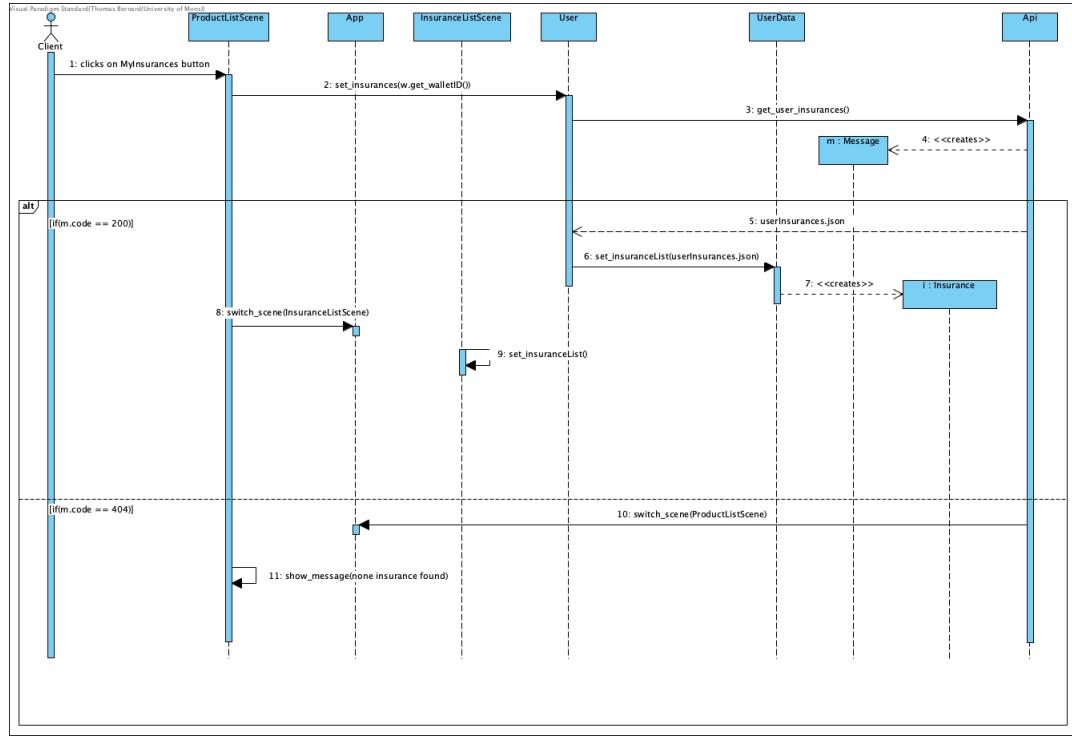


FIGURE 65 – Afficher la liste des assurances

Afficher la liste des assurances : Lorsque le client souhaite accéder à la liste de ses assurance une requête contenant le wallet et le userID du client est envoyée à l'API afin que celle-ci renvoie la liste des assurances. Il ya 2 réponses possible. Soit l'API trouve des assurances, met à jour la liste dans l'application Lorsque l'application change la scène pour afficher la InsuranceListScene la scène est instanciée et la liste est mise à jour à l'aide du fichier Json renvoyé par l'API. Sinon l'API renvoie une erreur. L'application change la scène sur la ProductListScene et informe le client qu'aucune assurance n'a été trouvée.

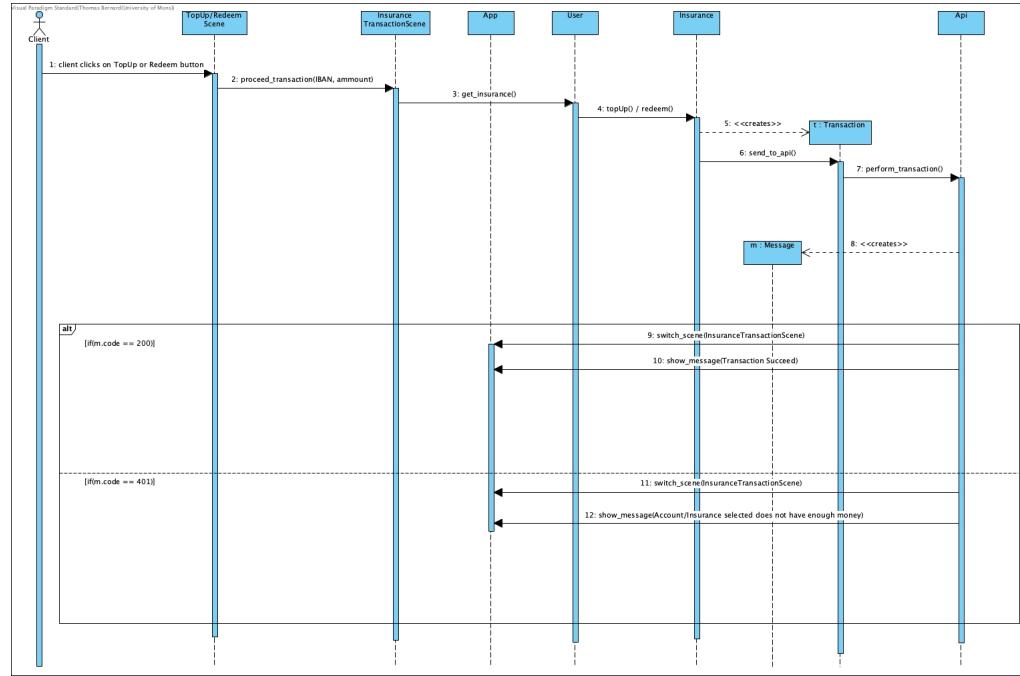


FIGURE 66 – Ajouter/retirer de l’argent d’une assurance

Ajouter/retirer de l’argent d’une assurance Lorsque le client souhaite ajouter/retirer de l’argent d’une assurance une requête est envoyée à l’API contenant le compte sur lequel il faut ajouter/retirer de l’argent ainsi que l’assurance courante. Il y a 2 types de réponses possibles. Soit tout se passe bien et l’API renvoie un message success à l’application. Dans ce cas l’application renvoie le client sur la InsuranceTransactionScene et met à jour les comptes ainsi que l’assurance.

Soit l’API renvoie un message d’erreur à l’application. Dans ce cas l’application renvoie le client sur la InsuranceTransactionScene et l’informe que le compte/assurance sélectionné pour l’ajout/retrait ne possède pas un solde suffisamment élevé et lui dit changer de compte ou de l’approvisionner avant de réitérer l’opération.

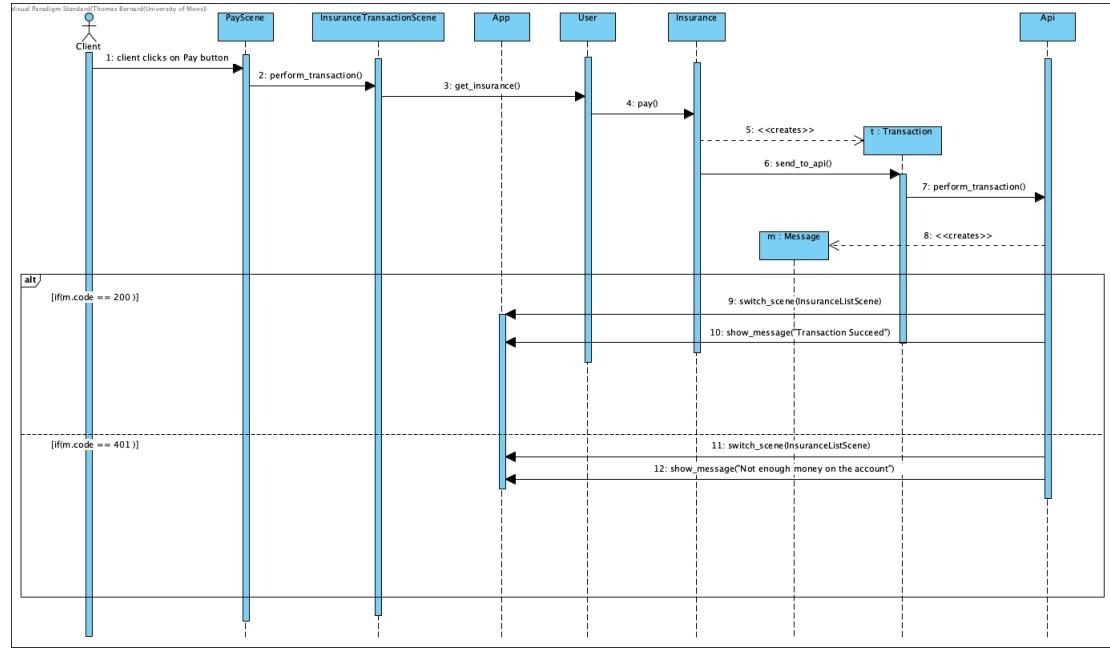


FIGURE 67 – Payer la prime de l’assurance

Payer la prime d’une assurance Lorsque le client souhaite payer la prime une requête contenant l’assurance en question ainsi que le montant et le compte à débiter est envoyée à l’API. Deux résultats sont possibles. Soit le solde du compte est suffisant et l’API renvoie un message de succès qui est affiché par l’application lorsqu’elle renvoie l’utilisateur dans la InsuranceListScene. Soit le solde est insuffisant et un message d’erreur est envoyé sur cette même scène à l’utilisateur lui demandé de sélectionner un autre compte ou bien de l’approvisionner.

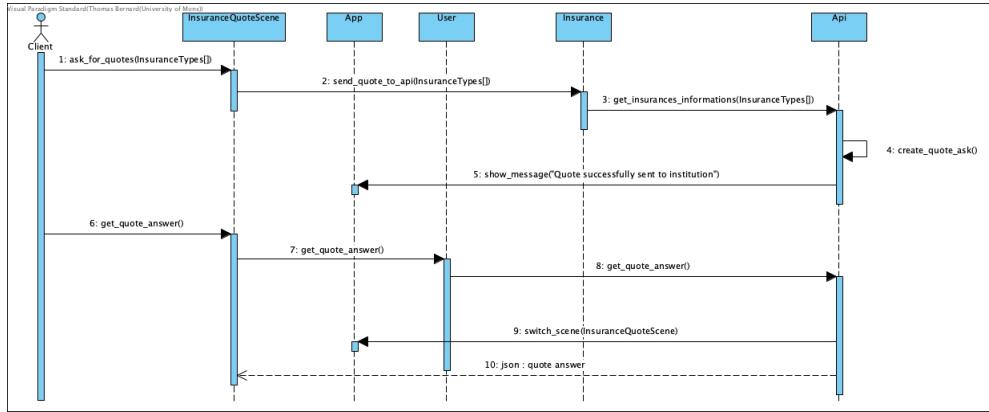


FIGURE 68 – Effectuer une demande de devis

Effectuer une demande de devis Lorsque le client effectue une demande de devis une requête contenant un type d'assurance est envoyée à l'API cette requête est traduite en un ajout dans la base de données par l'API qui ajoute la demande dans une table dédiée à cette utilisation. Ensuite l'API renvoie un message qui est affiché par l'application et qui informe l'utilisateur que sa demande a bien été envoyée à l'institution.

L'utilisateur peut également contrôler s'il a reçu une réponse dans ce cas on appelle une méthode dans l'API afin de contrôler s'il y a une réponse. L'API renvoie le devis au travers d'un fichier Json qui est interprété par l'application et affiché à l'utilisateur.

3.3.7 Application 2

3.3.8 Diagramme des cas d'utilisation

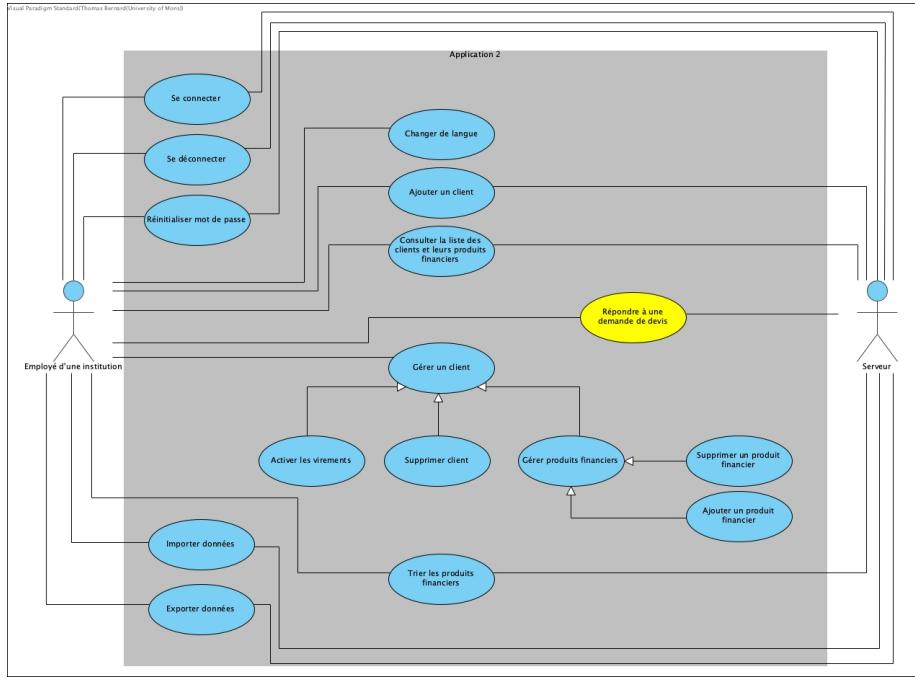


FIGURE 69 – Diagramme des cas d'utilisation de l'app 2 avec extension

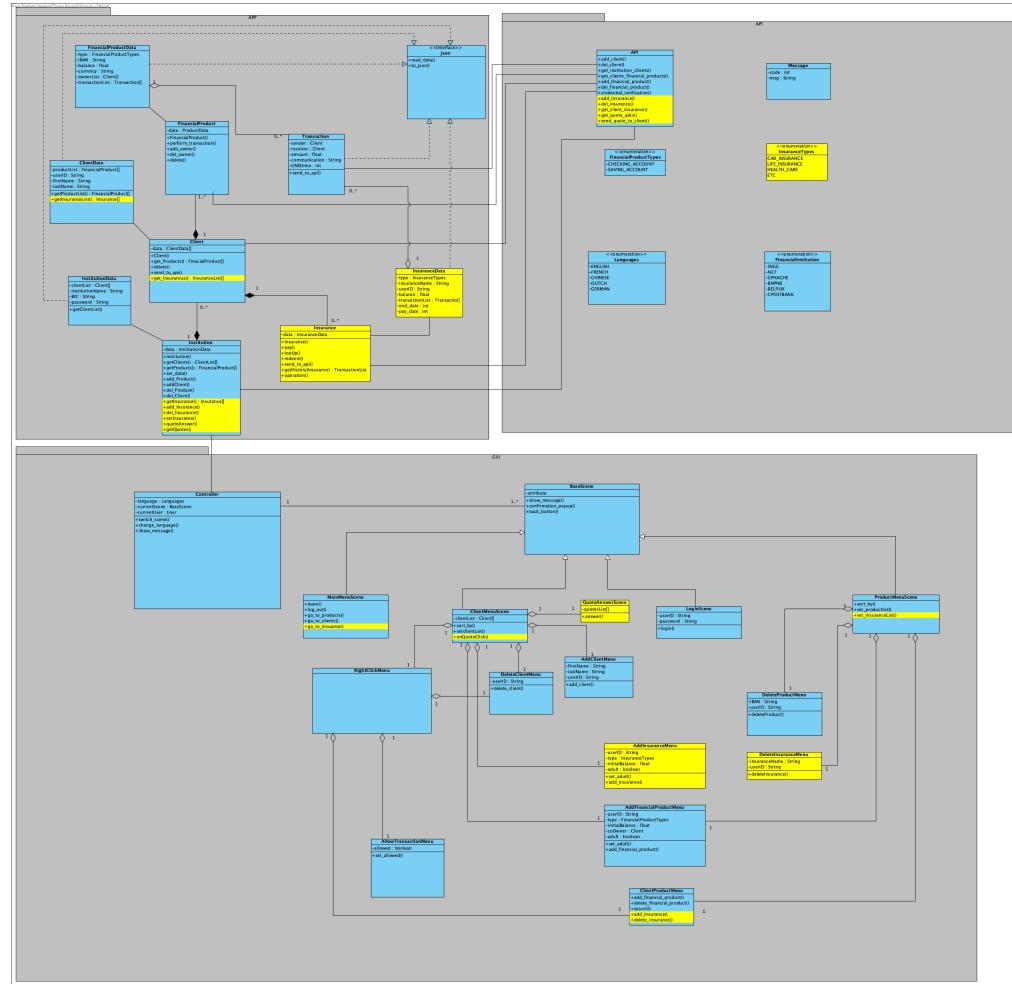
Le diagramme de cas d'utilisation de l'app 2 ne contient qu'une seule modification. Il s'agit du use case permettant de répondre à un devis. En effet lorsqu'une institution liste les produits d'un client elle liste également ses assurances. Il n'est donc pas nécessaire d'en rajouter plus.

3.3.9 Interaction Overview Diagram

Il n'y a également qu'un seul cas d'utilisation rajouté qui est celui de réponse à un devis. Par souci de taille du diagramme il n'a pas été ajouté au rapport. L'interaction a été rajoutée au niveau du menu de gestion des clients. Une fois l'utilisation terminée, le user de l'institution est ramené vers le menu des produits.

3.3.10 Diagramme de classes

Au niveau de la logique de l'application (package APP) mise à part le fait que la classe **Insurance** est maintenant reliée à la classe **Client** du fait que les institutions n'ont pas la notion de portefeuilles et qu'une méthode *quoteAnswer()*



ait été rajoutée à la classe **Institution** permettant d'envoyer le devis au client, il n'y a pas de changements importants par rapport à celui de l'application 1.

Au niveau de la partie serveur de l'application (package API) le seul changement à noter est l'ajout d'une méthode `answerQuote()` dans la classe **Api** qui permet à l'API de générer des réponses à une demande de devis.

Au niveau de la partie interface graphique de l'application (package GUI). Une méthode permettant d'accéder à la scène liée aux assurances a été ajoutée à la classe **MainMenuScene** et une méthode permettant d'accéder à la scène de demande des devis a été ajoutée dans la classe **ClientMenuScene**. Enfin une méthode permettant d'afficher les assurances des clients a été ajoutée à la classe **ProductMenuScene**.

Pour ce qui est des classes ajoutées, elles sont au nombre de 3.
Il s'agit des classes suivantes :

1. **QuoteAnswerScene** : qui est la scène où toutes les demandes de devis sont listées et qui contient un bouton permettant d'y répondre.
2. **AddInsuranceMenu** cette scène permet d'ajouter une assurance pour un client en particulier

3.3.11 Diagrammes de séquences

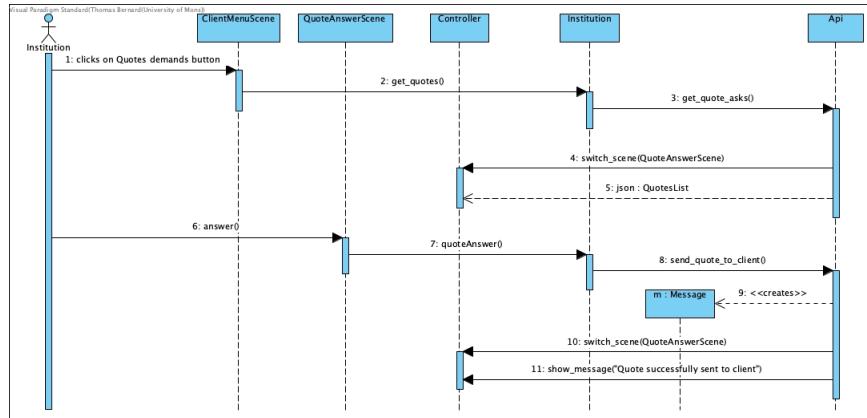


FIGURE 70 – Réponse à une demande de devis

L'insititution lorsque décide d'accéder à la scène des demande devis envoie une requête à l'API qui va rechercher la liste des demandes de devis qui sont propres à l'institution dans la base de données. Ainsi quand la scène est affichée la liste l'est également. Lorsque que l'insititution appuie sur answer une requête est envoyée à l'API qui va chercher les informations liées au type d'assurance demandé par le client et le stocke dans la base de données.

3.3.12 Serveur

3.3.13 Diagramme d'entité relation

Le diagramme d'entité relation a été étendu à l'aide de 3 tables : *INSURANCES*, *ISNURANCE_TYPE* et *INSURANCE_PRICE*.

La table *INSURANCES* contient toutes les assurances des clients. Elle peut être accédée par 2 tables : par la table *WALLETS* grâce au walletID qui lui est lié dans le cadre de l'application client et par la table *CLIENT_VS_INST* grâce au userID qui est lié à chaque assurances dans le cadre de l'application insititution.

La table *INSURANCE_TYPE* est une table intermédiaire contenant toutes les assurances proposées dans l'application. On peut obtenir le type d'une assurance depuis la table *INSURANCES* grâce au insuranceName.

La table *INSURANCE_PRICE* contient les informations pratiques liées à chaque type d'assurance. Car un type d'assurance peut avoir plusieurs nom (insName) notamment les assurances vies. Cette table est accessible depuis la table *INSURANCE_TYPE* grâce au insName.

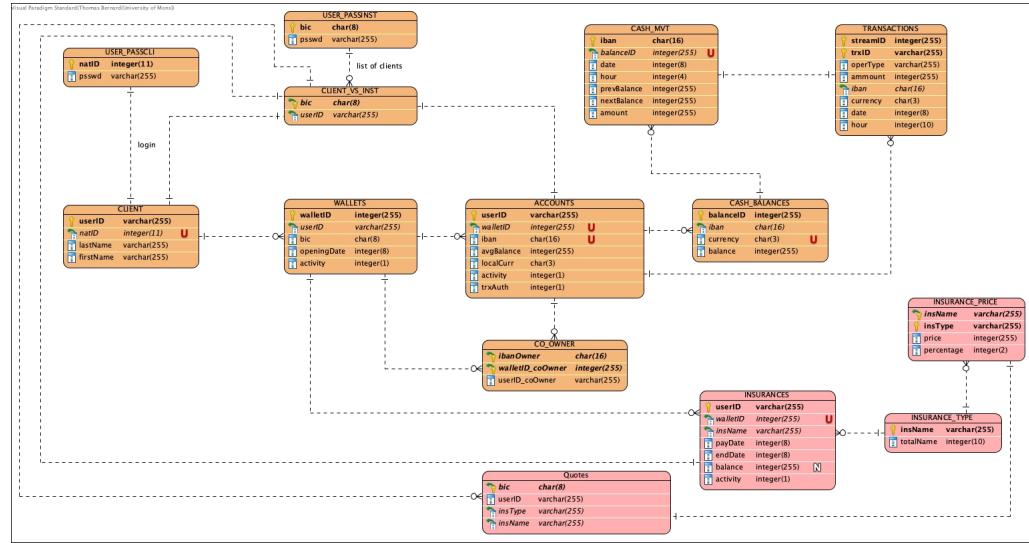


FIGURE 71 – Diagramme d'entité relation avec extension

3.3.14 Interface graphique de l'extension

Plusieurs scènes ont été rajoutées contenant les fonctionnalités de l'extension et des scènes de l'application de base ont été modifiées afin de donner accès à ces nouvelles scènes.

3.3.15 Application 1

1. NewInsuranceScene

Cette scène est la scène où l'on souscrit à une assurance elle est accessible

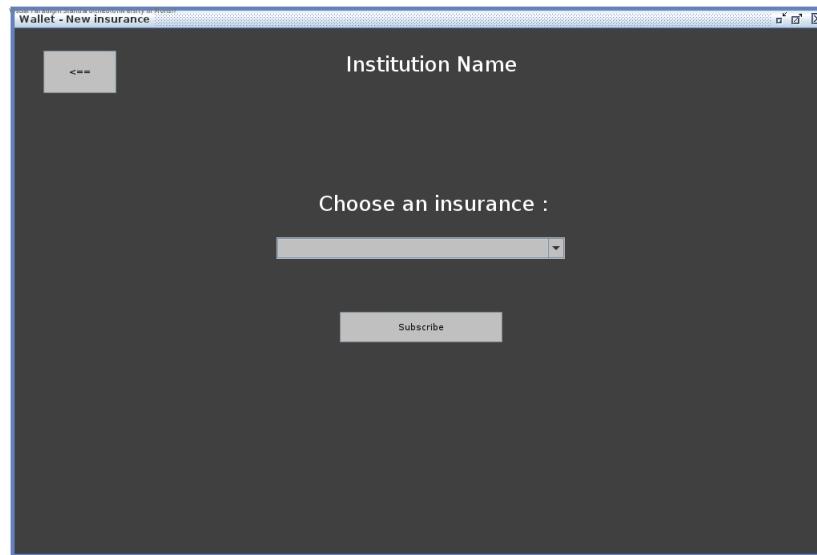


FIGURE 72 – Scène d'ajout d'une assurance

depuis la ProductListScene en cliquant sur le bouton "My Insurances"
Cette scène est composée d'une combo box permettant à l'utilisateur de choisir l'assurance à laquelle il veut s'inscrire ainsi qu'un bouton subscribe.

2. InsuranceMenuScene

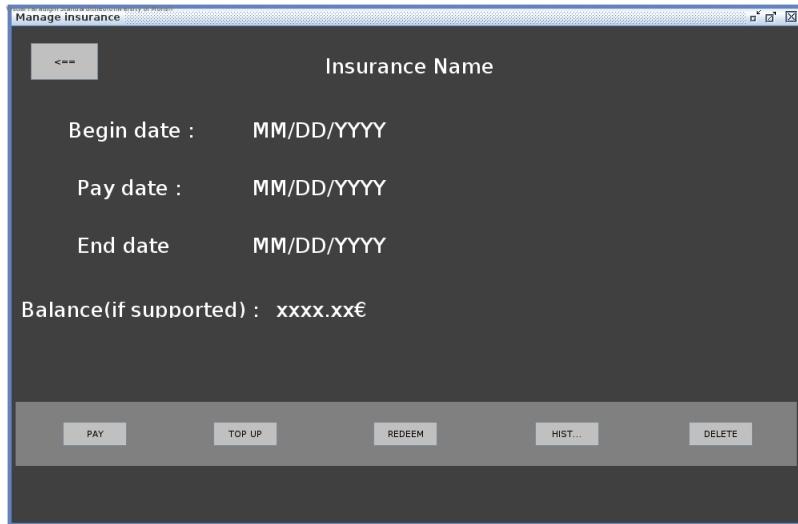


FIGURE 73 – Scène du menu d'une assurance

Cette correspond au menu d'une assurance. Toutes les informations pratiques de l'assurance y sont affichées ainsi que les actions que l'on peut effectuer sur celle-ci. Elle est accessible depuis l'InsuranceListScene en cliquant sur un bouton lié à une assurance.

Dans cette scène on retrouve 5 boutons. 3 permettant d'effectuer des transactions (PAY, TOP UP et REDEEM) ainsi qu'un bouton pour accéder à l'historique des transactions "HISTORY" et un bouton permettant de supprimer l'assurance "DELETE".

3. InsuranceListScene

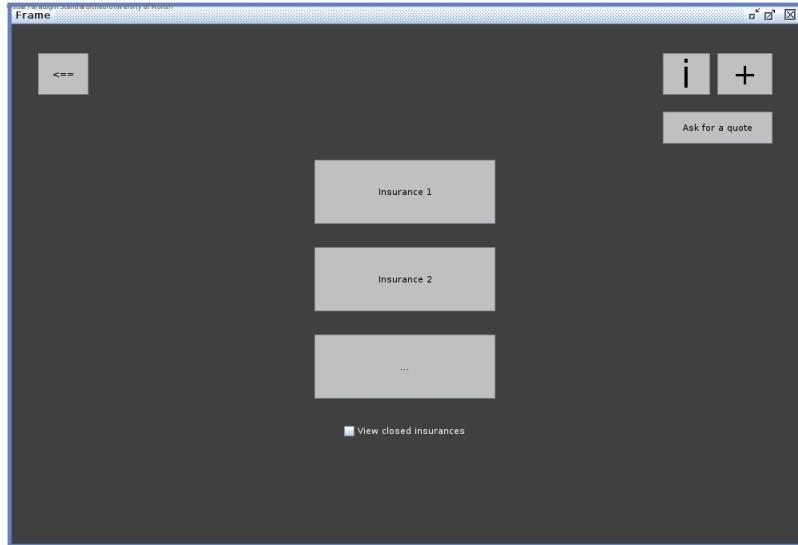


FIGURE 74 – Scène de la liste des assurances

Cette scène affiche la liste des assurances d'un client avec une check box "View closed insurances" permettant au client de voir les assurances qu'il a supprimé. Elle contient également les différentes assurances qui sont cliquable et redirige vers la InsuranceMenuScene. Il y a également un bouton "+" permettant de rajouter une nouvelle assurance et envoyant sur la NewInsuranceScene. Il y a un bouton "Ask for a quote" permettant de demander un devis et un bouton information "I" permettant d'obtenir des informations sur les assurances.

4. RedeemScene Cette scène affiche la demande de retrait d'argent il y a

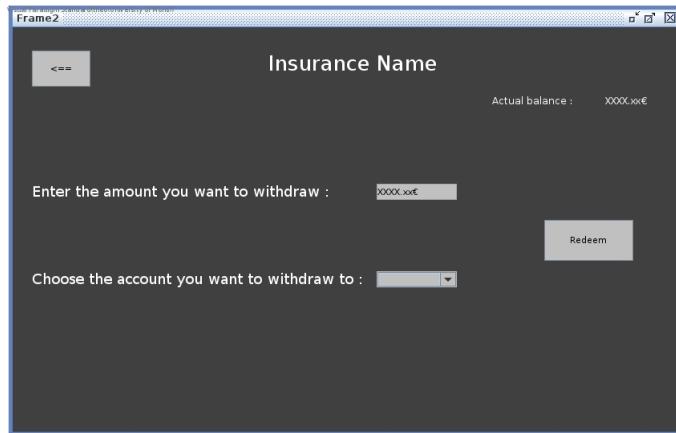


FIGURE 75 – Scène du retrait d'argent d'une assurance

1 textfield permettant d'entrer le montant à retirer et une combo box permettant de choisir le compte sur lequel on veut retier l'argent. Il y a un bouton "Redeem" permettant d'effectuer l'opération et un label affichant le solde actuel de l'assurance

5. TopUp Scene

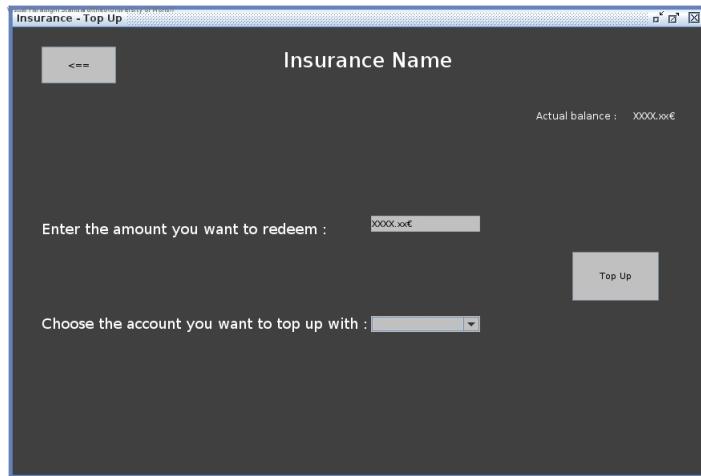


FIGURE 76 – Scène d'ajout d'argent d'une assurance
Cette scène fonctionne à peu de choses près comme la précédente. La scène PayScene n'a pas été représentée car jugée trop similaires à ces deux-ci.

3.3.16 Application 2

1. ClientsMenuScene

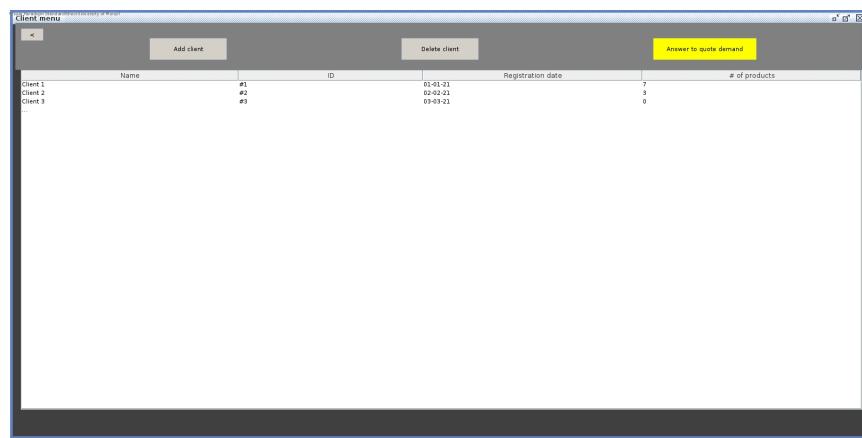


FIGURE 77 – Scène du menu des clients

Cette rajoute uniquement un bouton "Answer to a quote demand" permettant d'accéder à la scène de réponse aux devis.

2. QuoteAnswerScene

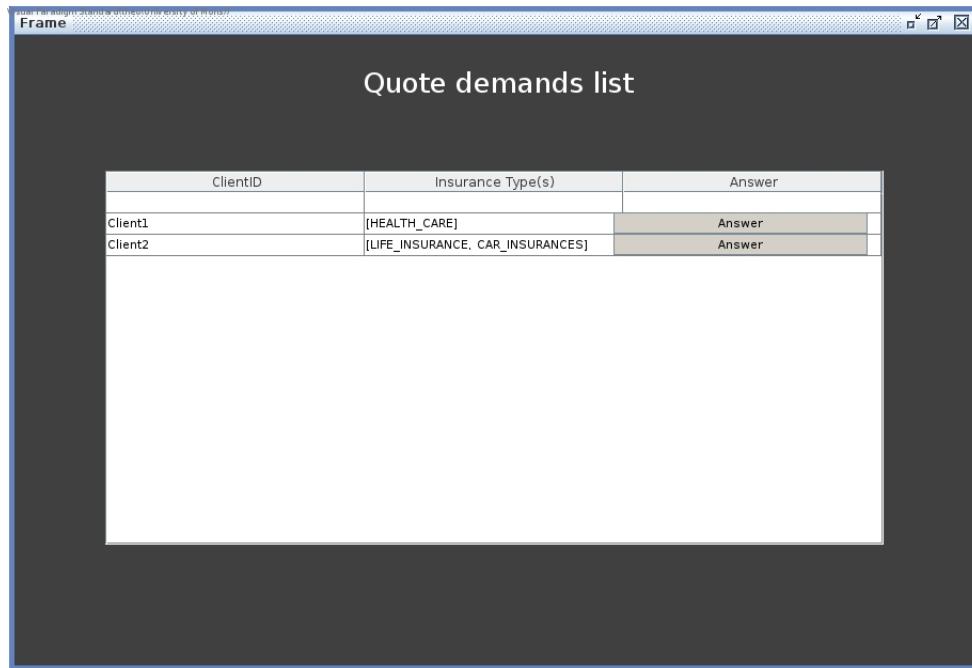


FIGURE 78 – Scène de réponse de demande à un devis

Cette affiche un tableau contenant les demandes des clients et un bouton "Answer" à coté de chaque afin d'envoyer la réponse au client.

3.4 Extension 6, Paiement et gestion des fraudes - Ag-benda Pignozi :

3.4.1 Vue d'ensemble

L'extension 6 Paiement et gestion des fraude a pour but de permettre à un client de générer des QRcode , de scanner des QRcode (évènement simulé en pratique),

effectuer des payments avec et sans contact(évènement simulé en pratique) et d'avoir un système de détection de fraud permettant de restreindre ou dans certains cas de bloquer des transactions suspecte.

Pour l'instution cette extension implement un système de limites de transactions portant sur les payments avec et sans contact.

Le plus gros des ajouts sont effectué pour l'application 1 et donc les diagramme se basent fortement sur celle-ci.

L'application 2 se vois n'ayant que la définitions de limites de payments, elle est quasiment identique a celle de base.

Les ajouts effectué au diagrammes des 2 applications sont démarqué par une couleur rose.

3.4.2 Application 1

3.4.3 Diagramme des cas d'utilisation

Pour ce diagramme, j'ai rajouté toutes les nouvelles actions qu'un client pouvais effectuer.

"Effectuer Transaction" qui comprends "Effectuer un payment avec contact " , "Effectuer un payment sans contact"

et "Payment par code QR" les 3 étant des specialized usecase de "Effectuer une transaction".

J'ai aussi ajouté "Generer un code QR" et enfin la généralisation " Gestion de black list" avec les spécialisations

"ajouter à la blacklist" et " retirer de la blacklist " qui représentent le fait que l'utilisateur peut maintenant définir et modifier une liste de compte à bloquer.

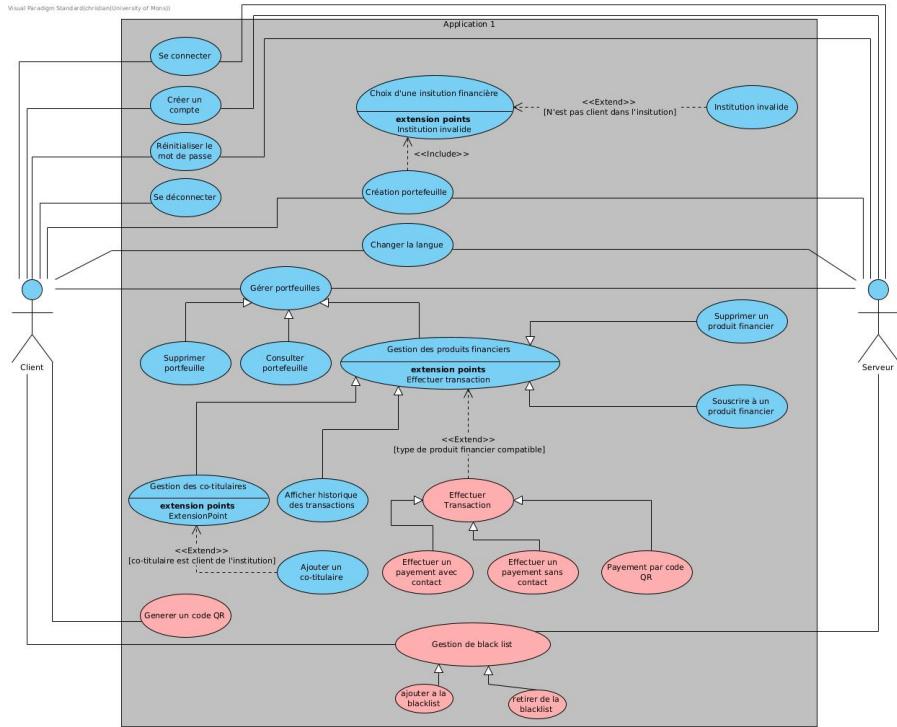


FIGURE 79 – Diagramme des cas d'utilisation de l'app 1 avec extension

3.4.4 Interaction Overview Diagram

Ce diagramme se base sur celui de l'application 1.

3.4.5 Diagrammes de classe

Dans le package APP de l'application il ne figure q'une seule nouvelle class : **QRcode**. Elle contient les attributs suivant : *image* qui est une objet de type *Image* qui est l'image du code QR. *amount* qui est une objet de type *float* qui est le montant de la transaction. *IBAN* qui est une objet de type *String* qui est le compte du receveur. *creationTime* qui est une objet de type *int* qui est l'heure et la date de création du QRcode en UNIXTime. *communication* qui est une objet de type *String* qui est une communication (optionel) de la transaction. contient également les méthodes liées aux assurances. La classe **Insurance** est *Transaction* que vous avez ajouté 2 nouveaux attributs : *place* qui est un objet de type *String* qui est l'endroit du payment, et *country* qui est de type *String* et qui est le pays de destination du virement. Dans le package API je n'ai rajouté que *timecheck* qui est une méthode permettant de vérifier la l'heure et la date de création d'un QRcode.

Dans le package GUI j'ai rajouté les nouvelles scènes dont l'utilisateur a besoin pour : Générer un QRcode, scanner un QRcode, effectuer un payment sans/avec contact,gérer sa blacklist.

3.4.6 Diagrammes de séquence

Effectuer une payment avec contact : Lorsque le client veux effectuer une payment sans contact il doit tout d'abord sélectionner le payment ensuite il choisit le mode de paiement avec contact. Après avoir entré le bon code PIN , la requête de la transaction est envoyée à l'API qui va vérifier la validité. Si la transaction est suspecte alors elle est soit annulé ou mise en attente de confirmation en fonction du niveau de suspicion. Si la transaction n'a pas été effectué le client est notifié avec un message. La procédure est identique pour le payment sans contact hormis le fait qu'en cas de faible suspicion , le mot de passe de l'utilisateur est demandé.

Generer un QRcode : Le client peut générer un QRcode et elle se stocke localement. Il a pour ça besoin de rentrer le compte sur le quel il souhaite recevoir l'argent, le montant et une communication eventuelle. Scanner un QRcode consiste à charger un QRcode et d'effectuer une transaction à partir de ses informations.

3.4.7 Application 2

3.4.8 Interaction Overview Diagram

Il n'y a q'un seul cas rajouté a ce diagramme. Ce cas réfère a la gestion de limites sur les payments avec et sans contact.

3.4.9 Diagramme de classes

Dans le package App j'ai rajouté des attributs référant au diffèrent types de limites qui peuvent être mise sur les virement avec ou sans contact. L'objet *Institution* possède désormais une méthode permettant de définir les limites des virment avec ou sans contact nomée *set_limits*.

L'API contiens maintenant des méthodes permettant d'envoyer *set_limits* et de recevoir *get_limits* du serveur.

Dans le package GUI j'ai rajouté la scène permettant aux employés des instituions de définir les plafonds de transactions.

3.4.10 Diagrammes de séquences

Si l'employé d'une institution a besoin de définir des plafonds sur les virement avec ou sans contact il peut le faire en effectuant un clique droit sur le menu des clients. Grace à ça il a maintenant accès à différentes options parmis lesquelles se trouve la définitions de limits. En cliquant dessus , il pourra entrer les différentes valeurs des limites souhaitées et confirmer celles-ci. Après confirmation, la GUI au travers de la méthode *et_limits()* d'Institution va envoyer ces limites à l'API.

3.4.11 Serveur

3.4.12 Diagramme d'entité relation

Le diagramme d'entité relation a été étendu à l'aide de 8 attribut dans la table *ACCOUNT* et de 5 attributs dans la table *TRANSACTION*

Les nouveaux attributs de la table *ACCOUNT* sont les limites qui peuvent être définis pour les virements avec ou sans contact. Les nouveaux attributs de la table *TRANSACTION* servent à la détection de fraude.

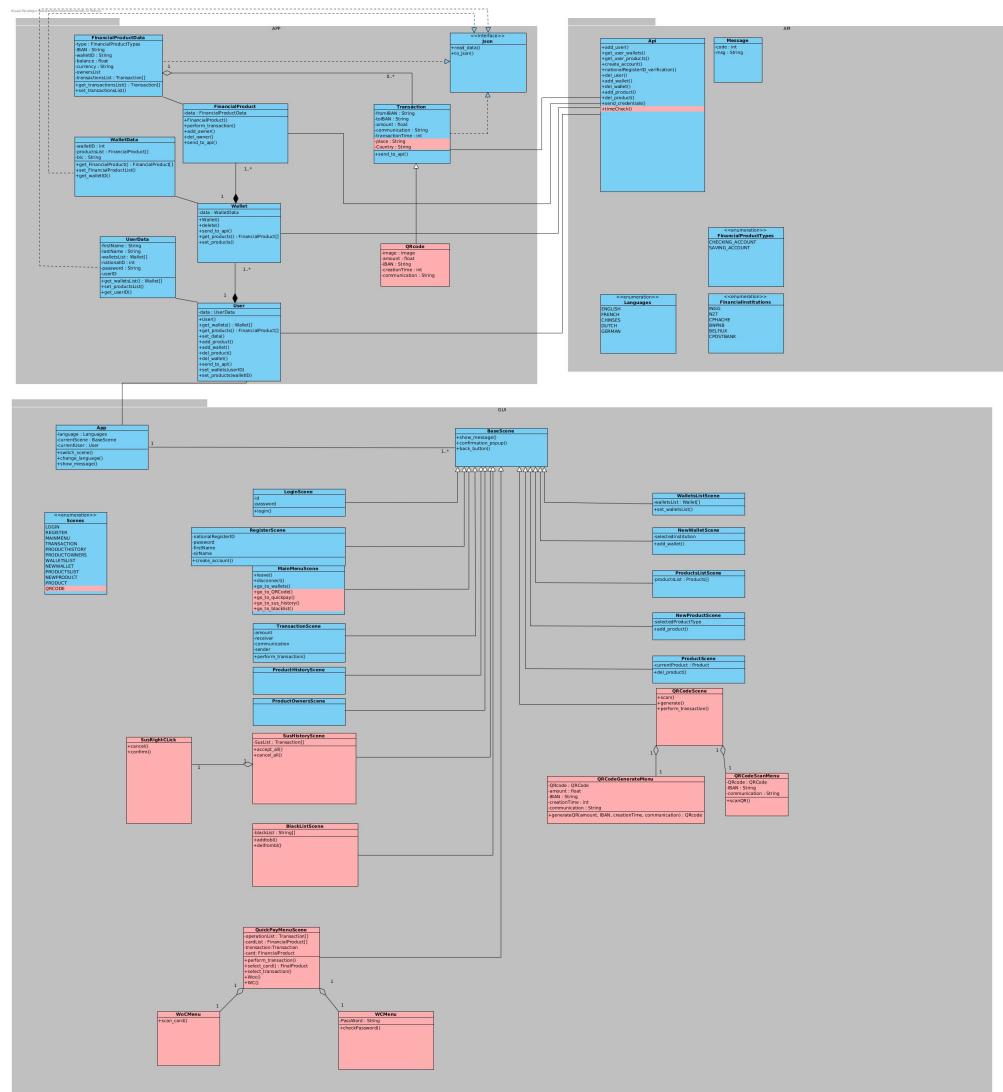


FIGURE 80 – Diagramme de classes de l'app 1 avec extension

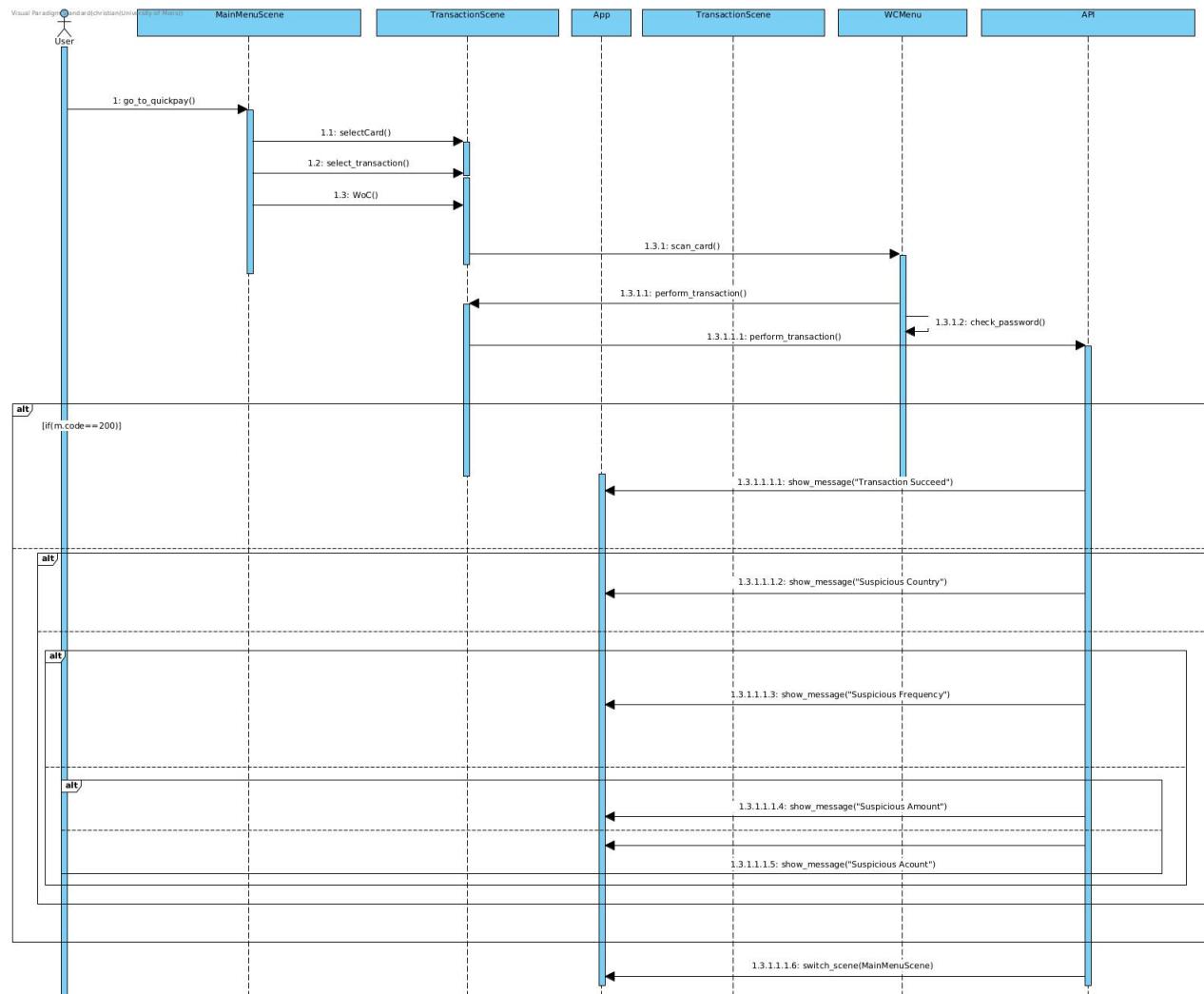


FIGURE 81 – Effectuer un paiement avec contact

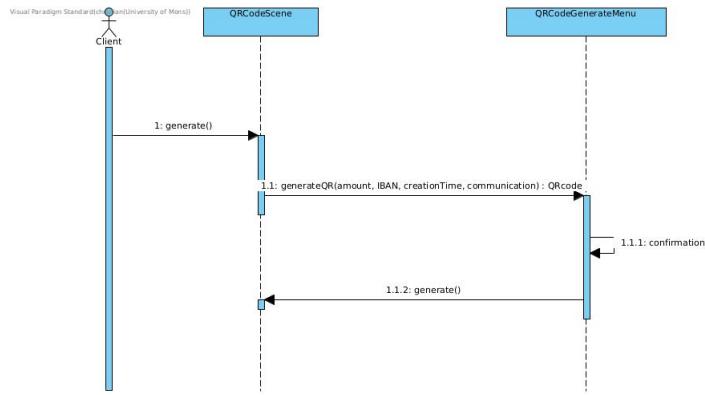


FIGURE 82 – Génère un QRcode

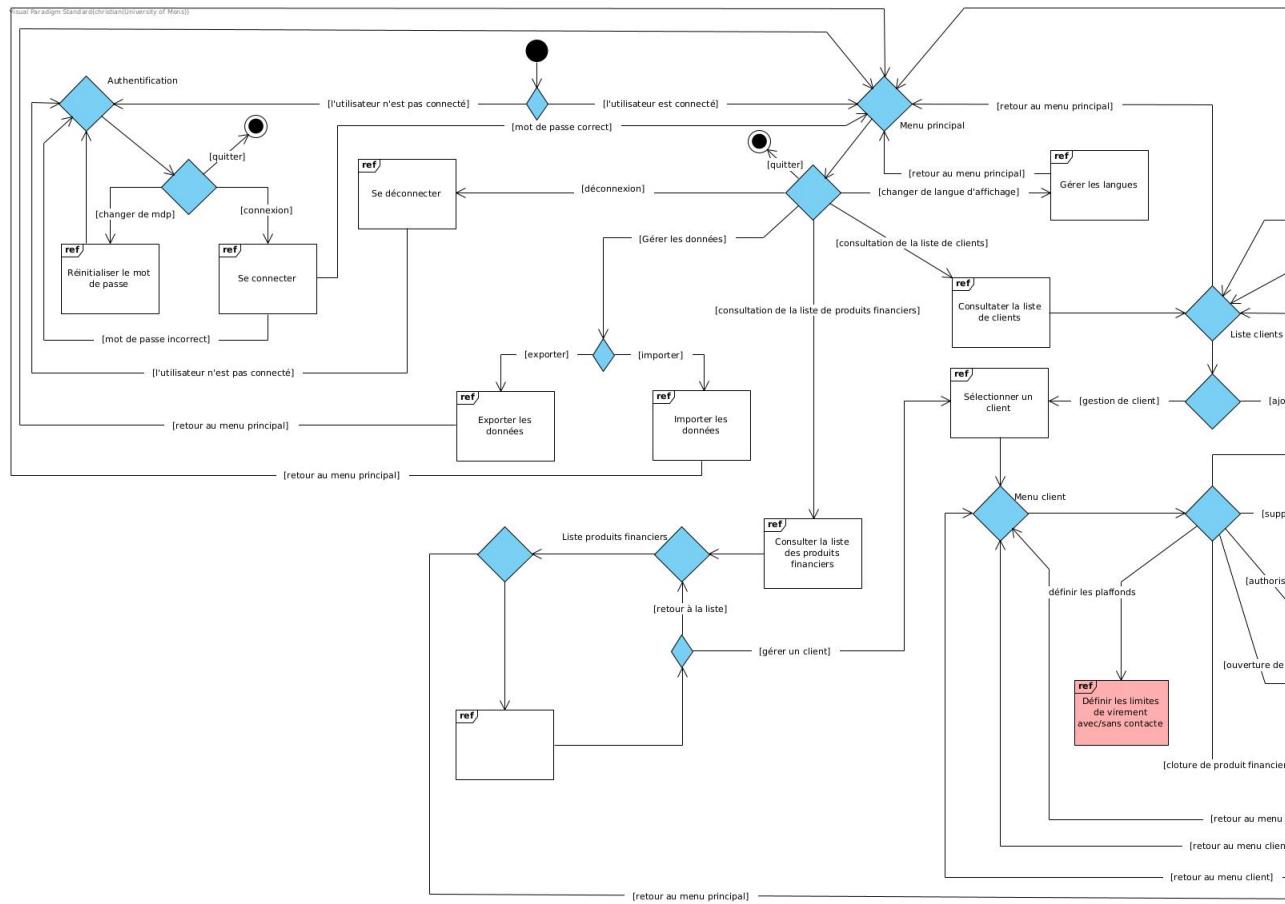
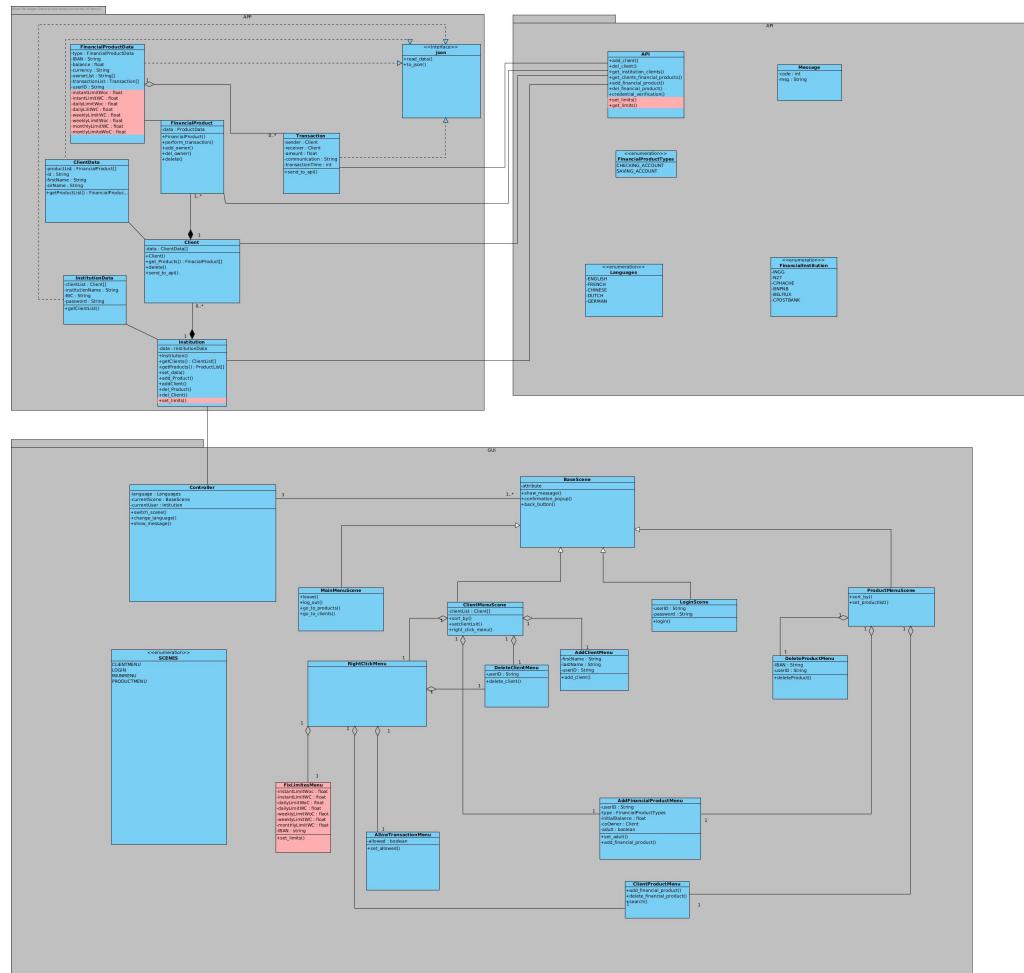


FIGURE 83 – Interaction Overview App2



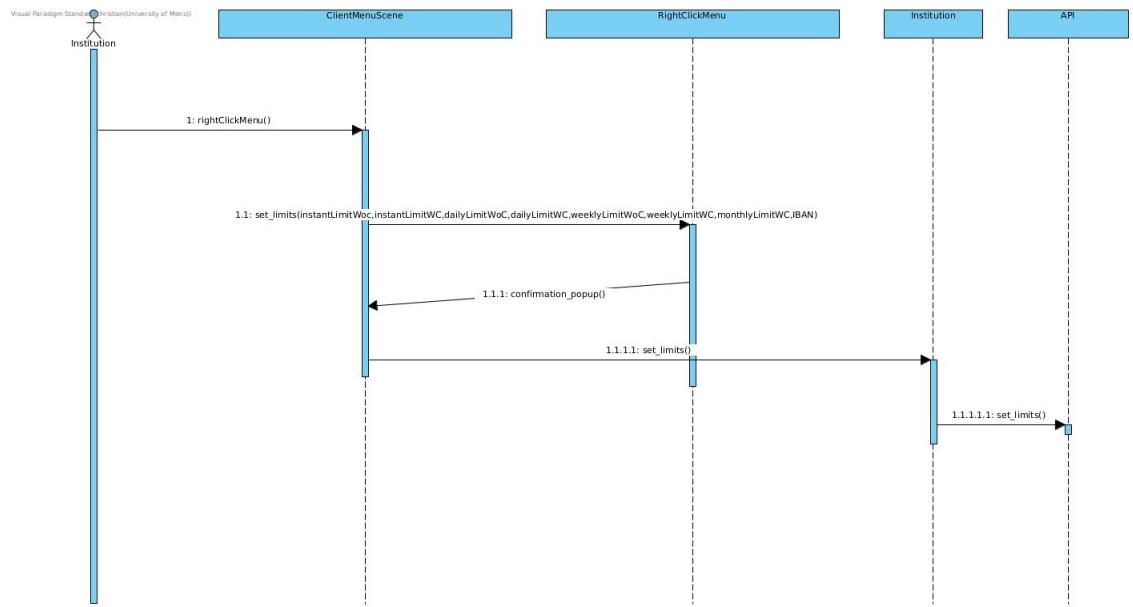


FIGURE 84 – Définir les plafonds sur les virements avec ou sans contact

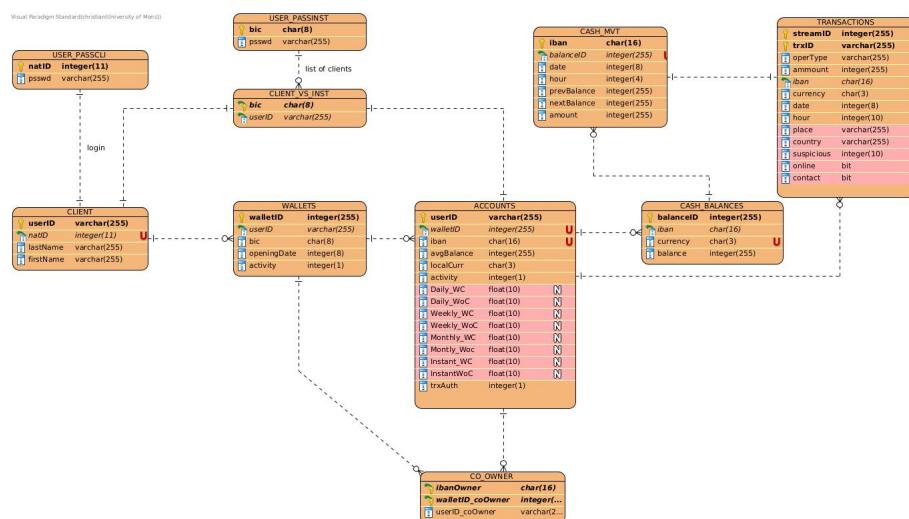


FIGURE 85 – Diagramme d'entité relation avec extension