TP : Analyse et Conception Objet de Logiciel

Alix BRUCKERT
Adrien DELEPLACE
Paul DUDNIC
Valérian THOMAS

Mars-Avril 2020 - IF 2A

Contents

1 Cahier des charges

1.1 Présentation de l'équipe de développeurs

L'équipe de développeurs est composée de 4 élèves de Grenoble-INP Ensimag en deuxième année.

1.2 Présentation du client : la banque

XXX Banque et Assurance demande un logiciel dédié à ses clients permettant de réaliser des consultations et transactions.

1.3 Produit désiré : Logiciel client pour gestion de comptes bancaires

1.3.1 Fonctionnalités

Le logiciel devra permettre à l'utilisateur de :

- Se connecter : Les utilisateurs peuvent se connecter à l'outil à l'aide d'un login et d'un mot de passe. Un compte comporte également une adresse électronique valide qui permet l'envoi de différents courriers;
- Accéder au descriptif de ses différents comptes : Les utilisateurs peuvent consulter à tout moment les différents soldes de leurs comptes, ainsi que l'historique des dépenses et des revenus de chaque jour. Il est aussi possible fixer des plafonds par compte;
- L'utilisateur peut ajouter une étiquette aux dépenses pour ainsi les catégoriser. Il sera possible d'afficher les statistiques de dépense d'un compte bancaire;
- Voir l'état de ses transactions: L'utilisateur peut savoir l'état des différentes transactions d'un compte, à savoir en cours ou traitée;
- Effectuer un virement: Les utilisateurs peuvent effecteur un virement bancaire entre deux comptes. L'un doit forcément être un compte personnel, l'autre soit un compte personnel, soit un compte extérieur. L'utilisateur a donc la possibilité d'ajouter un compte bénéficiaire pour ses virements. L'utilisateur a aussi la possibilité d'effectuer des virements réguliers automatiques;
- L'utilisateur peut voir les cartes possédées, et faire opposition.
- Gérer ses emprunts et épargnes : Les utilisateurs peuvent consulter à tout moment le contrat de l'emprunt ainsi que les mensualités de celui-ci;
- Se déconnecter de son espace personnel : Les utilisateurs peuvent se déconnecter en toute sécurité de leur espace personnel. Lors du retour sur l'application, l'utilisateur devra à nouveau rentrer son identifiant et son mot de passe afin d'accéder à ses comptes et ses informations;

1.3.2 Structure du logiciel

Pour ce logiciel, nous utilisons une structure Modèle-Vue-Contrôleur(MVC). Cette structure permet entre autre d'englober l'ensemble de nos classes représentant des objets dans une partie de Modèle, tandis que l'interface graphique et les boutons serviront de Vue et de Contrôleur.

1.4 Le livrable

1.4.1 Charte Graphique

Le client n'impose aucune charte graphique précise, mais exige que le livrable soit bien présenté, clair et facile à utiliser, y compris par des personnes non familiarisées à l'outil informatique.

De plus, son logiciel doit s'adapter à différents types de support, à savoir le téléphone mobile, l'ordinateur et la tablette mobile.

1.4.2 Contraintes techniques

Le code fourni devra impérativement être en Java ou C++ au choix de l'équipe de développeurs.

1.4.3 Limites du logiciel

La création de compte n'est pas prise en charge par l'application.
Aucun service de messagerie n'est prévu pour communiquer avec le banquier.

1.4.4 Planning

L'ébauche du logiciel et toute sa phase d'analyse et de conception sont à remettre au client pour le 14 avril 2020.

1.4.5 Capacités

L'application devra pouvoir être utilisée par plusieurs milliers de clients en simultané, et être performante.
L'application devra être rapide (par plus de 2 secondes de chargement).

2 Document d'analyse

2.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation représente les actions réalisées par le système pour avoir un résultat qui répond au besoin d'un acteur particulier. Nous ne considérons ici que le diagramme des cas d'utilisation de l'utilisateur de l'application du point de vue client (de la banque).

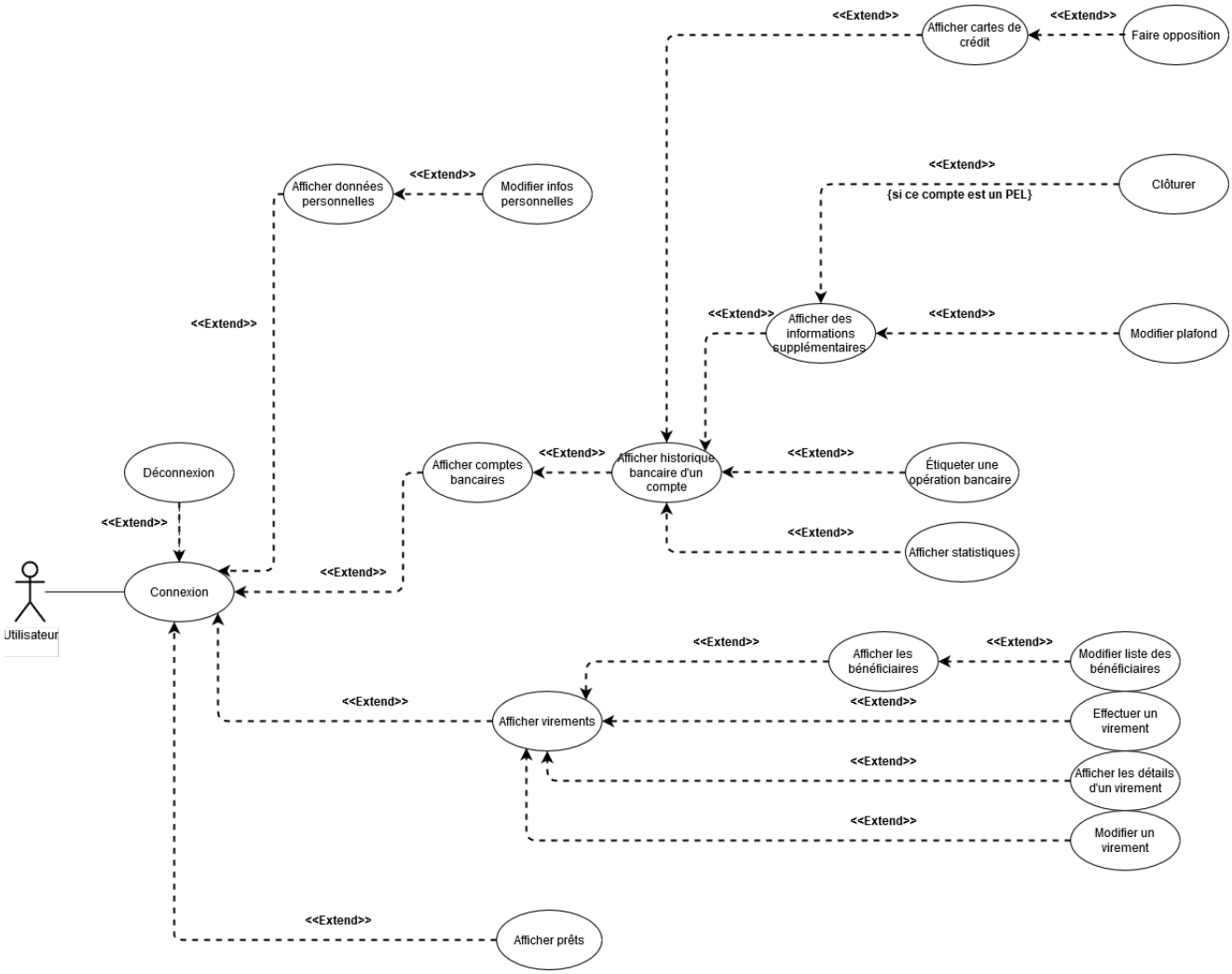


Figure 1: Diagramme des cas d'utilisation

2.2 Cas d'utilisation

L'utilisateur sera capable au travers de l'application de réaliser la liste non exhaustive des actions suivantes :

- Afficher le statut d'une des cartes d'un compte Courant;
- Bloquer une carte;
- Afficher les informations d'un compte comme l'IBAN;
- Réaliser des transactions entre un compte où les transactions sont permises, et un autre compte, Bénéficiaire ou personnel;
- Consulter les données d'un Emprunt comme le montant restant, le taux ou encore l'échéancier;

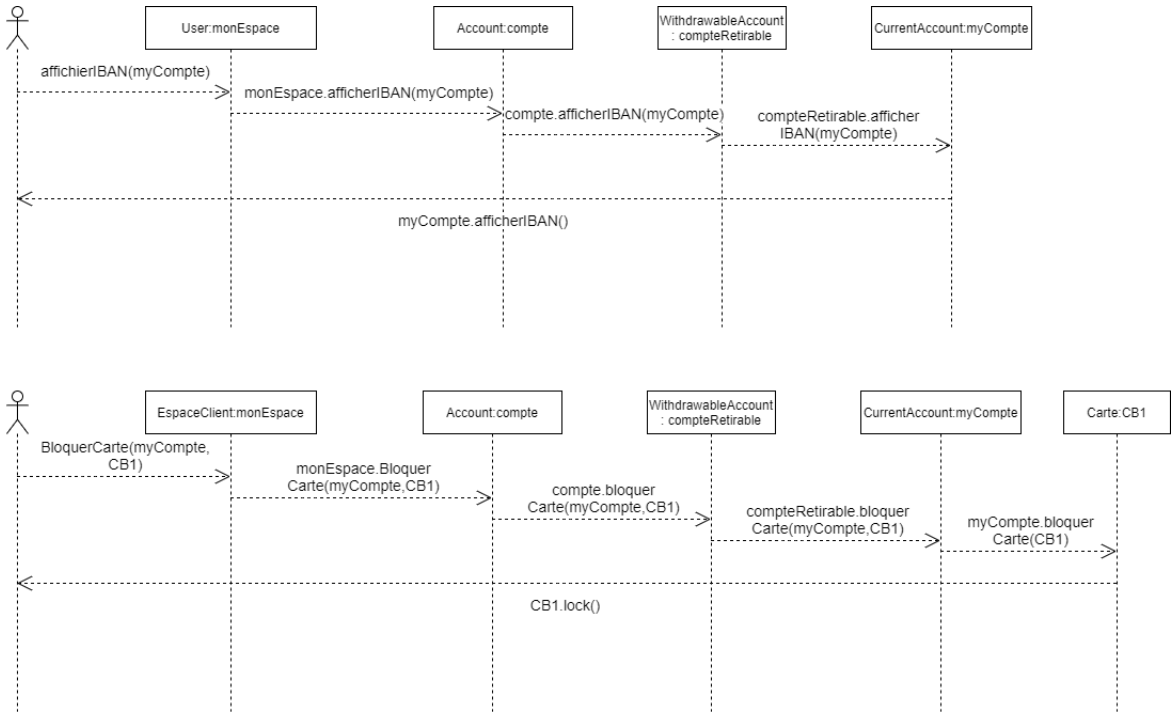


Figure 2: Diagramme Séquence : Affichage IBAN et Blocage de carte

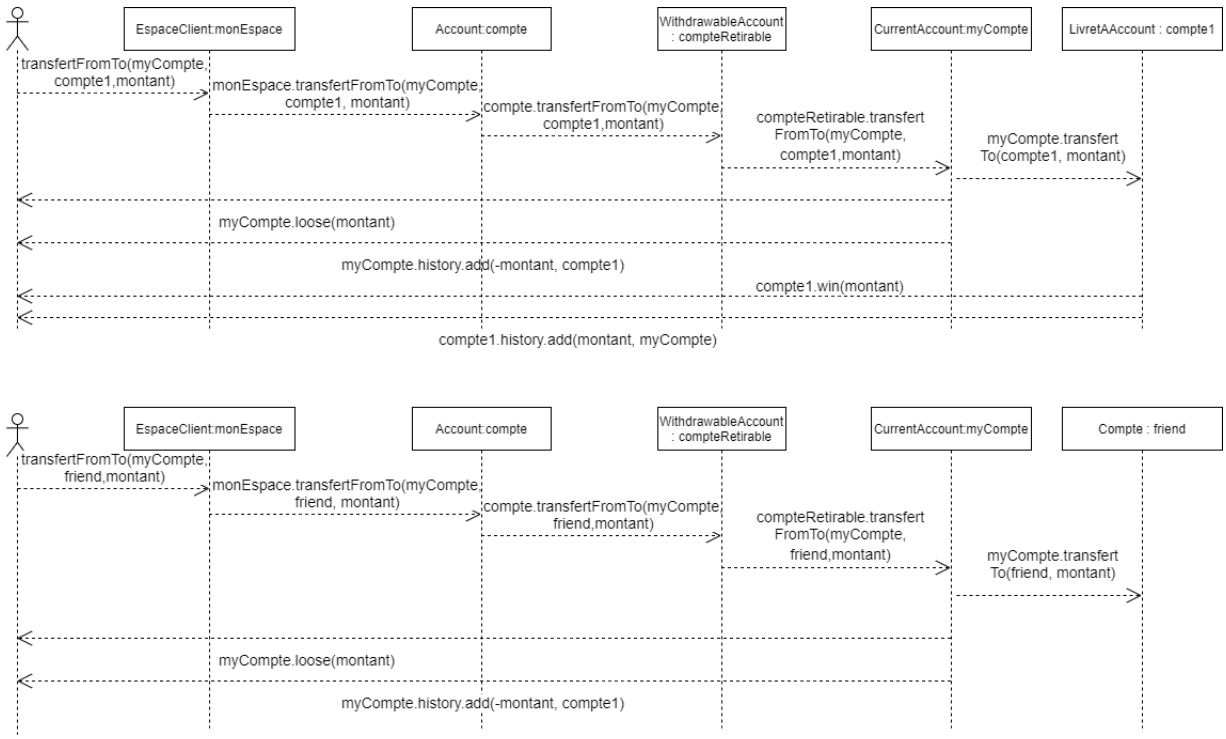


Figure 3: Diagramme Séquence : Réalisation d'une transaction avec un compte Personnel et un Bénéficiaire

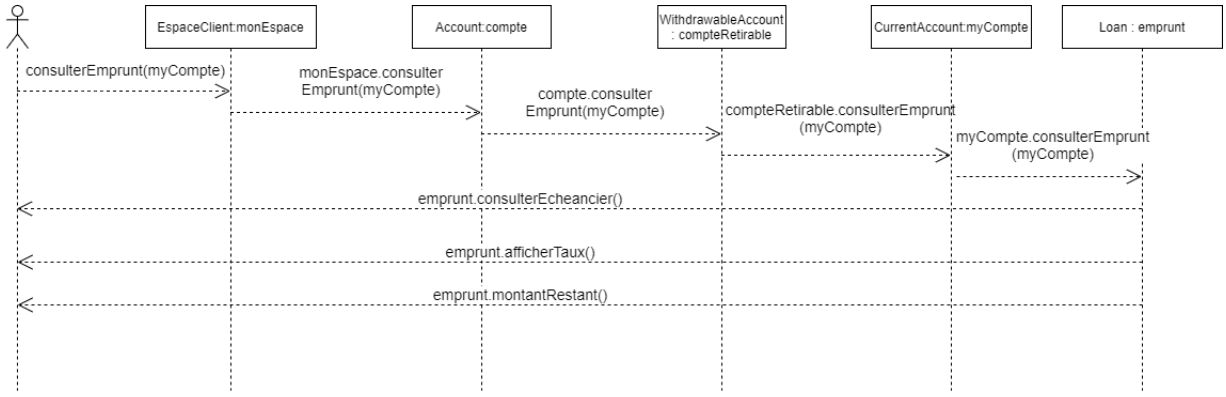


Figure 4: Diagramme Séquence : Affichage du montant restant à rembourser

2.3 Diagramme des classes d'analyse

L'application contient un panel de classes qui ont servi à sa conception. Ces classes sont associées entre-elles selon l'utilisation de chacune. Le diagramme

de classe d'analyse ci-dessous permet faire le détail :

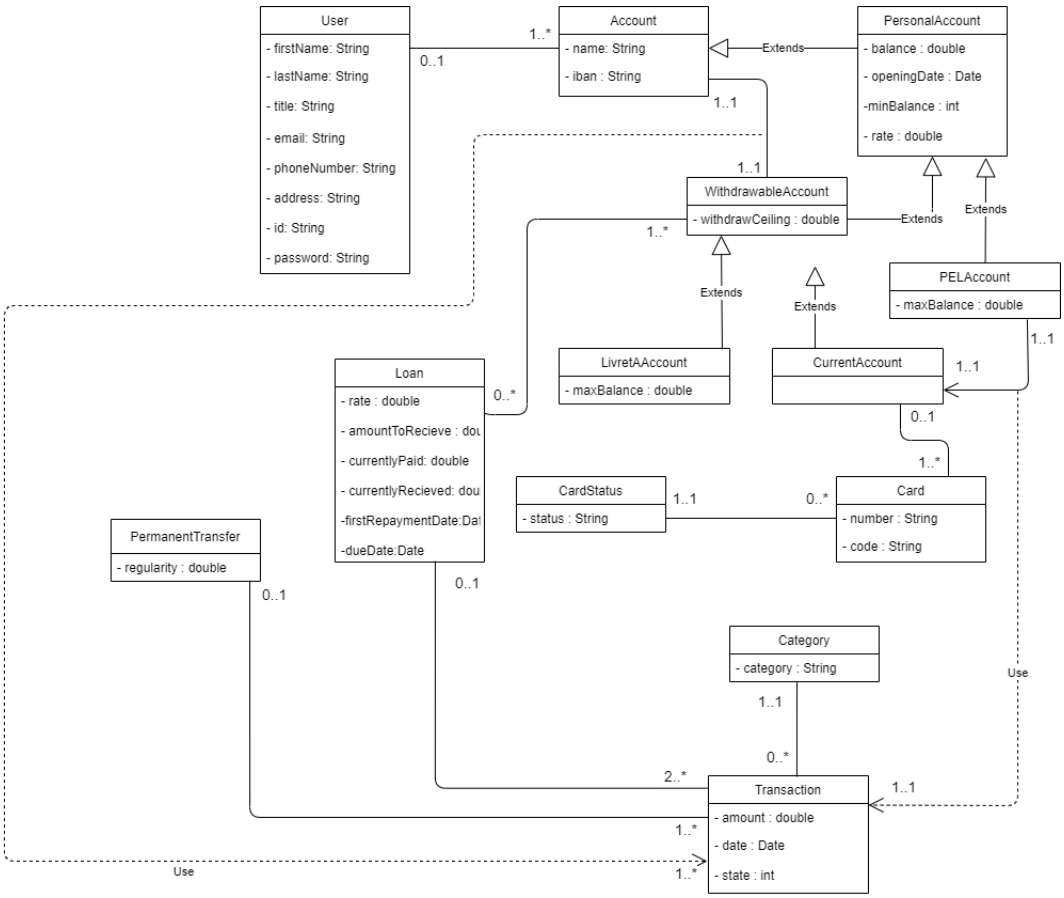


Figure 5: Diagramme de classe d'analyse

- Une classe *User* représentant l'espace client d'un utilisateur. Cet espace est donc associé à tous les comptes du client;
- Une classe *Account* représentant les comptes personnels et les bénéficiaires disponibles du client;
- Une classe PersonalAccount qui représente les Comptes bancaires qui appartiennent à l'utilisateur.
- Une classe WithdrawableAccount, qui représente les comptes à partir desquels il est possible de faire une transaction (sans les PEL par exemple).

- *PELAccount* qui est un compte personnel sur lequel on ne peut faire des transactions mise à part l'action de "casser" le compte, auquel cas il y a une transaction entre le compte *PEL* et un compte *Courant* d'un montant égal au solde du PEL;
- *LivretAAccount* qui est un compte personnel sur lequel les transactions sont possibles mais sans une carte bancaire associée. De plus, des transactions entre ces comptes et un compte de la classe *Account* sont possibles;
- *CurrentAccount* qui représente les comptes courants du client, associé à une ou plusieurs cartes bancaires. De plus, des transactions entre ces comptes et un compte de la classe *Account* sont possibles;
- Une classe *Card* représentant la ou les cartes d'un compte bancaire, associé avec le statut de la carte ("bloquée" ou "valide");
- *Transaction* représente une transaction entre 2 comptes (l'un étant forcément un compte personnel). Cette transaction possède une méthode de comparaison et est relié à un historique, une catégorie et un statut ("exécutée" ou "en cours").
- *PermanentTransfer* représente des transactions réalisées de manière régulière dans le temps, comme un loyer ou un des prélèvements mensuels/trimestriels etc..
- *Loan* représente un emprunt que l'on peut consulter. Cet emprunt est donc relié à 2 transactions régulières : l'emprunt et son remboursement, et est donc relié à 2 transactions ou plus.

2.4 Diagrammes d'états-transitions

Cette section détaille l'utilisation de notre application sous forme schématique : une case représente un État, tandis qu'une flèche symbolise une transition (= action de saisie ou de clic par exemple).
Le premier schéma est le diagramme global, et les diagrammes suivants sont des zooms sur certaines parties du diagramme global (références au "voir 1", "voir 2", "voir 3" et "voir 4" du premier diagramme).

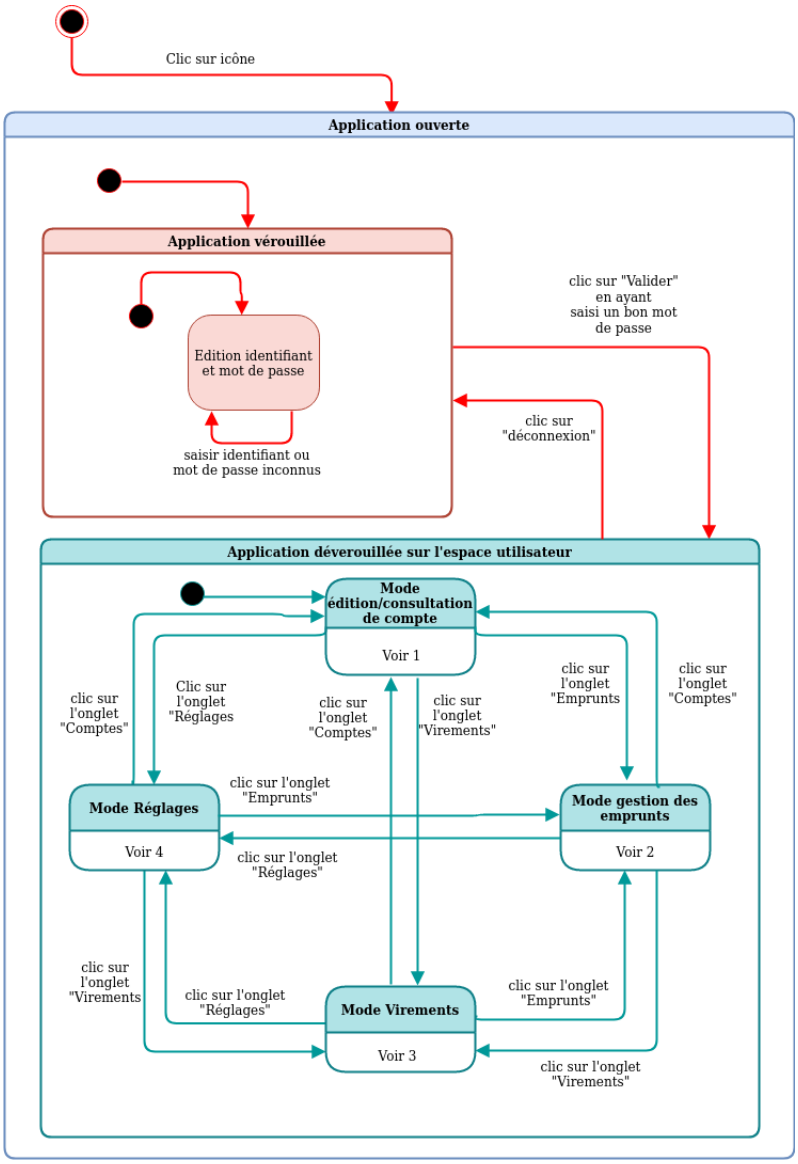


Figure 6: Schéma États-Transitions global

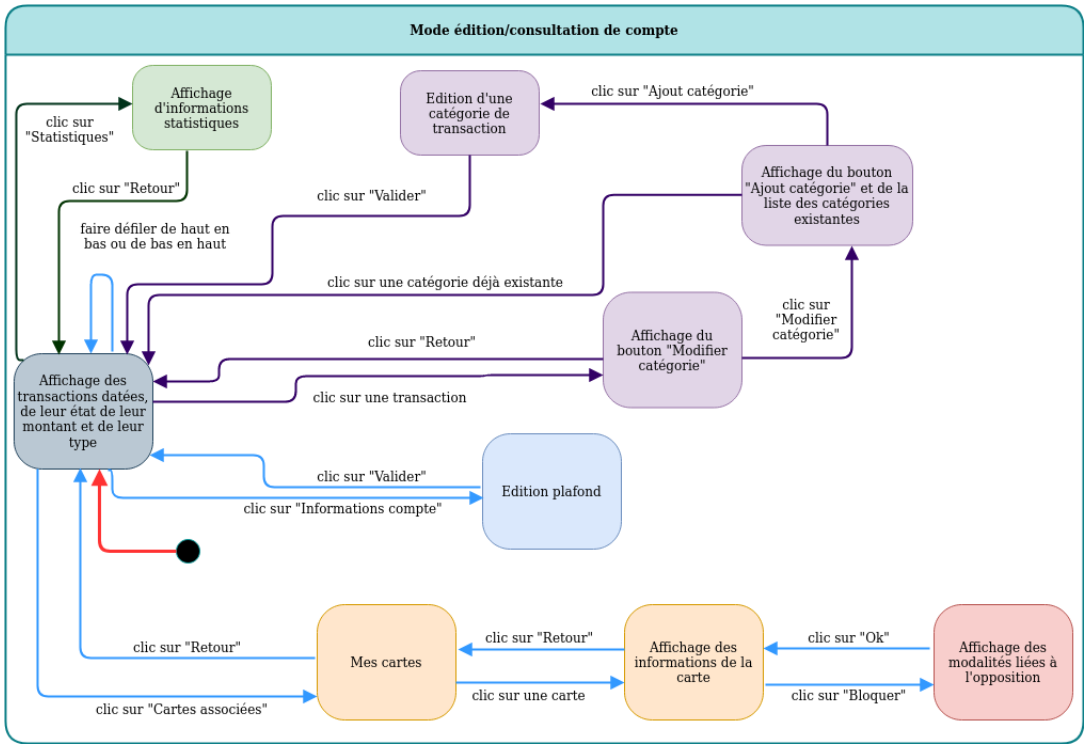


Figure 7: Schéma États-Transitions Gestion/Consultation compte (*voir 1)

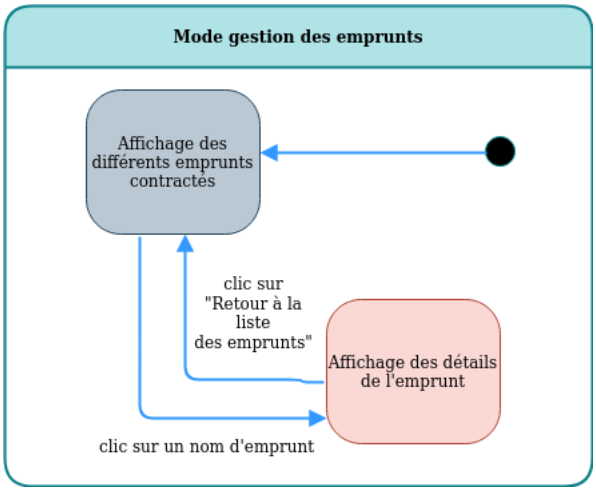


Figure 8: Schéma États-Transitions Gestion des emprunts (*voir 2)

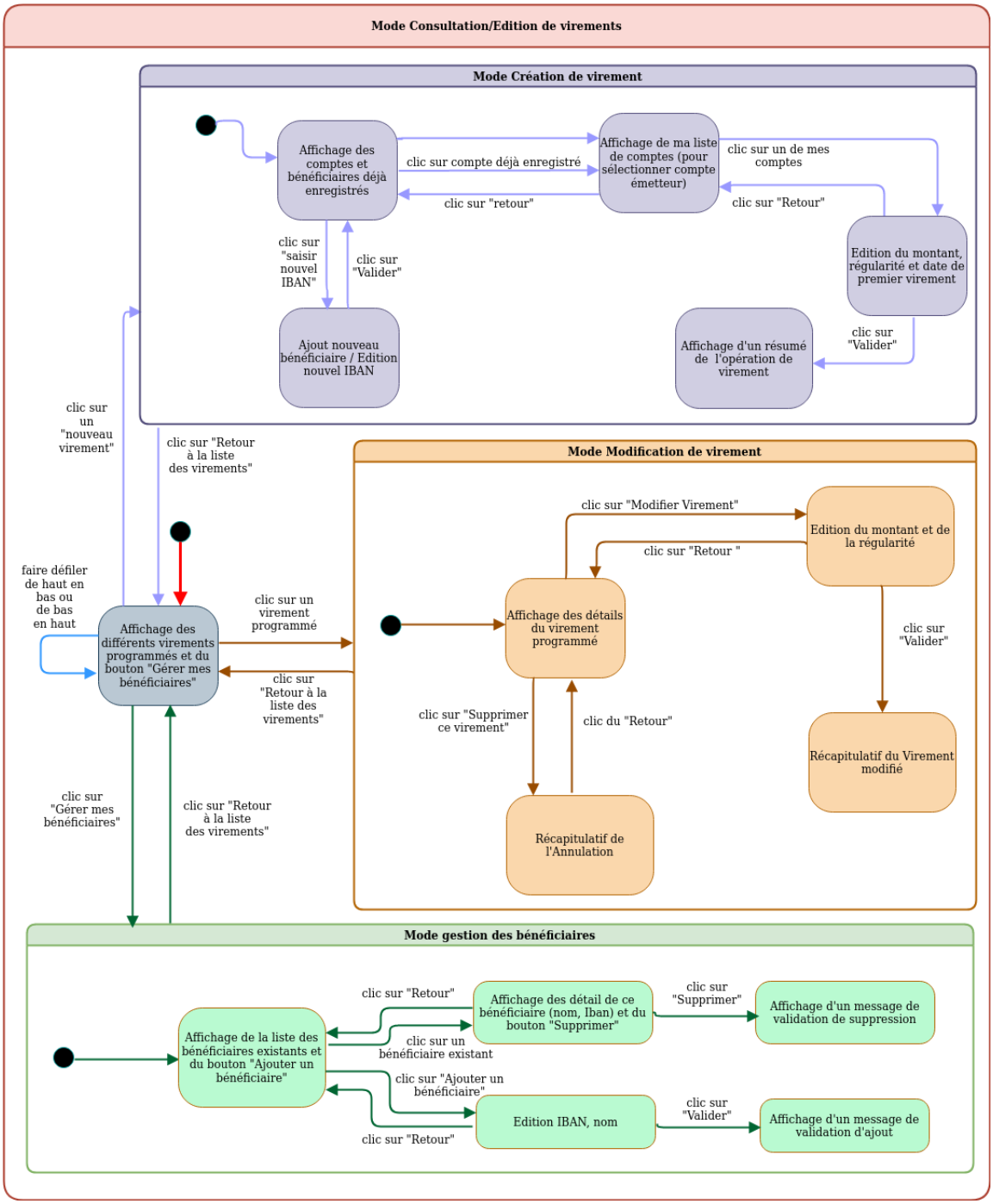


Figure 9: Schéma États-Transitions Gestion des virements (*voir 3)

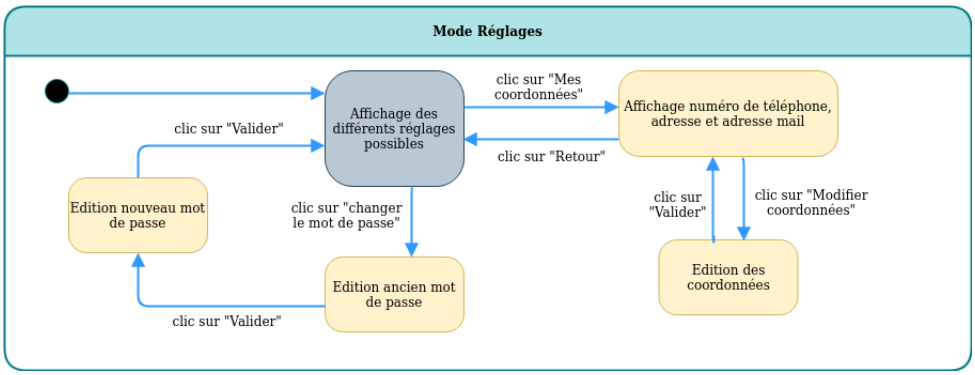


Figure 10: Schéma États-Transitions Réglages (*voir 4)

3 Document de conception

3.1 Architecture logique du logiciel

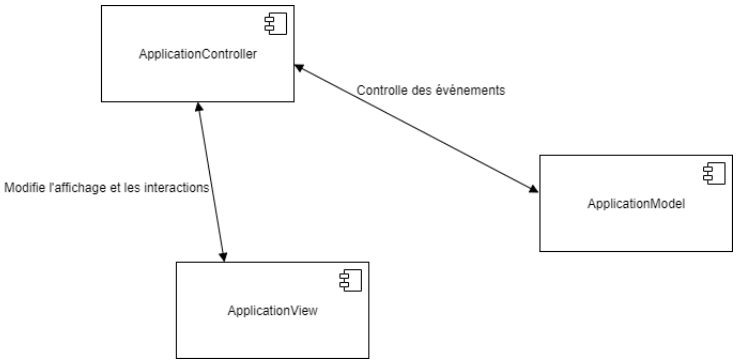


Figure 11: Diagramme de Conception en Modèle-Vue-Contrôleur

Voir le descriptif détaillé des graphes View et controler se trouvent dans l'annexe aux pages 18 et 19.

Nous utilisons une architecture *Modèle - Vue - Contrôleur*, qui est tout à fait adaptée à l'utilisation d'interfaces graphiques. Le contrôleur gère les entrées de l'utilisateur et met à jour le modèle. L'affichage est également mis à jour par le contrôleur lors des changements de vues, et à chaque modification du modèle. Nous aurions pu utiliser le pattern Observer pour mettre à jour l'affichage, mais il était plus simple ici de passer par le contrôleur. Ceci nous permet de garder le modèle indépendant de la vue, mais aussi la vue du modèle.

3.2 L'incrément choisi

L'ensemble de l'application bancaire est codée en JAVA (jdk 1.8) avec les classes et fonctionnalités des packages SWING et AWT en particulier de java.
L'ensemble des diagrammes et des graphes sont faits sur draw.io en ligne.

Notre application n'est évidemment pas tout à fait complète, comparée à une réelle application bancaire : par exemple, nous avons créé trois types de compte (courant, livret A, PEL) alors qu'il en existe d'autres différents. Autre exemple, notre application ne peut afficher sous forme de pdf les contrats mis en place. Il est aussi impossible d'avoir des statistiques concernant les catégories des transactions.

De même, nous n'utilisons pas de base de donnée dans notre incrément, contrairement à une vrai application. Ici pour simuler le fonctionnement de notre application dans une situation réelle (avec une base de données), nous utilisons un jeu de données initialisées au lancement de l'application. Cela permet d'interagir avec l'application, mais les modifications ne sont pas enregistrées et elles sont réinitialisées à chaque fois que l'application est ouverte.

3.3 Conception

Voir descriptif détaillé du diagramme de classe de la partie Modèle dans l'annexe à la page 20.

4 Manuel Utilisateur

Ce manuel a pour vocation de décrire les actions que peut effectuer l'utilisateur de l'application. On commence par la connexion à son espace client et on termine pas la déconnexion et la fermeture de l'application.

4.1 Connexion - déconnexion

- ME CONNECTER A L'APPLICATION
  - J'écris mon mon de passe et mon identifiant dans les espaces prévus à cet effet.
  - Je clique sur le bouton de validation "login". Si j'ai saisi un identifiant et mot de passe valides, alors un message de bienvenue m'annonce que je suis connecté, et en cliquant sur "Ok", j'arrive sur mon espace client.
- ME DÉCONNECTER DE L'APPLICATION
  - Je clique sur l'onglet "Déconnexion".
  - J'arrive sur la fenêtre de saisie du login et mot de passe.
- FERMER L'APPLICATION
  Je clique sur l'onglet "Quitter".

4.2 Réglages

- CHANGER MON MOT DE PASSE
  - Je clique sur l'onglet "Réglages".
  - En bas de la page, je clique sur "changer code secret".
  - Je saisis deux fois mon nouveau code, et clique sur "confirmer". Si, je change d'avis avant de confirmer, alors je clique sur "Annuler".

- CHANGER OU CONSULTER MES COORDONNÉES
  - Je clique sur l'onglet "Réglages", et je vois s'afficher mes coordonnées actuelles. A côté de chaque coordonnée, un bouton "changer".
  - Si je clique sur un bouton "changer", je saisis deux fois ma nouvelle coordonnée et je clique sur "confirmer".
  - Si, je change d'avis avant de confirmer, alors je clique sur "Annuler".

4.3 Mes comptes

- ACCÉDER AUX INFORMATIONS SUR MES COMPTES : informations générales, historique transactions, cartes associées
  - Je clique sur l'onglet "Comptes".
  - S'affiche la liste de mes comptes avec leur nom, solde, type et montant total des opérations en traitement.
  - Si je veux accéder aux informations sur un compte en particulier, je clique sur ce compte.
  - S'affiche alors l'historique de mes transactions.
  - Si je veux accéder aux informations générales sur mon compte, je clique sur "Informations compte" en dessous de l'historique des transactions. S'affichent alors les plafonds, date d'ouverture, nom du titulaire, IBAN.
  - Si je veux voir quelles cartes de paiement sont liées à ce compte, je clique sur "Cartes associées" en dessous de l'historique de mes transactions. S'affiche alors la liste de toutes les cartes associées à ce compte sous forme de liste déroulante.

- MES CARTES DE PAIEMENT
  Une carte est associée à un compte. Donc pour accéder à une carte, il faut d'abord aller sur le compte auquel elle appartient.
  - Je clique sur l'onglet "Comptes".
  - Je clique sur un compte dans la liste qui s'affiche.
  - Je clique sur "Cartes associées". S'affiche la liste des cartes associées à ce compte avec leur numéro, nom du titulaire, état (Active/Bloquée).
  - Pour faire opposition à une carte, cliquer sur "Bloquer", cliquer sur "Ok", saisir de nouveau mon mot de passe et valider.

- GÉRER LES PLAFONDS DE MES COMPTES
  - Je clique sur l'onglet "Comptes".
  - Je clique sur le compte qui m'intéresse dans la liste déroulante qui s'affiche.

- Je clique sur "Informations compte".
- Si je veux modifier le plafond de mon compte, je clique sur "Modifier", je saisi alors le montant, je clique sur "Confirmer", je saisis mon mot de passe, et je clique sur "Confirmer".

- ÉTIQUETER UNE TRANSACTION
  - Je clique sur l'onglet "Comptes".
  - Je clique sur le compte qui m'intéresse dans la liste déroulante qui s'affiche.
  - S'affiche l'historique de mes transactions sous forme de liste déroulante.
  - Je clique sur la transaction à étiqueter.
  - Je choisis dans le menu une transaction existante oubien "Ajouter étiquette", puis je clique sur "Ok". - Si j'ai sélectionné une étiquette existante à l'étape précédente, je saisis mon mot de passe et je valide. - Si j'ai sélectionné "Ajouter étiquette" à l'étape précédente, je saisis mon étiquette, je valide, je saisis mon mot de passe, puis je valide à nouveau.

4.4 Mes emprunts

- ACCÉDER A MES EMPRUNTS
  Je clique sur l'onglet "Emprunts".
  Je peux alors accéder à l'ensemble des informations de l'emprunt.

4.5 Mes virements

- CONSULTER MES VIREMENTS RÉGULIERS
  Je clique sur l'onglet "Virements". S'affiche alors la liste de mes virements programmés réguliers sous forme de liste déroulante.

- PROGRAMMER UN VIREMENT
  - Je clique sur l'onglet "Virements".
  - En bas de la page, je clique sur "Nouveau virement".
  - Je sélectionne le compte émetteur dans le premier menu déroulant et je valide.
  - Pour le compte receveur, je peux le sélectionner de la même façon dans le 2e menu déroulant si ce dernier fait déjà partie de mes bénéficiaires. Sinon, je clique sur "Ajout bénéficiaires", je saisis son nom et son IBAN puis je clique sur "Confirmer".
  - Je saisis le montant de mon virement.
  - Je coche si je veux que mon virement soit de type permanent.
  - Si je n'ai pas coché le type permanent, je n'ai plus qu'à valider, saisir mon mot de passe et valider à nouveau.
  - Si j'ai coché le type permanent, alors je sélectionne dans le menu déroulant la régularité, puis je saisis la date jj/mm/AA du premier virement. Ensuite, je valide, je saisis mon mot de passe, et je valide à nouveau.

- MODIFIER UN VIREMENT
  - Je clique sur l'onglet "Virements".
  - Je clique sur le virement que je veux modifier.
  - Je peux supprimer le virement en cliquant sur "Supprimer ce virement".
  - Je peux modifier le montant du virement en cliquant sur "Éditer montant". Je saisis deux fois mon nouveau montant et clique sur "Confirmer" pour valider la modification.
  - je peux modifier la régularité du virement en cliquant sur "Éditer régularité". Je choisis dans le menu déroulant ma nouvelle régularité et clique sur "Ok" pour valider la modification.

- GÉRER MES BÉNÉFICIAIRES
  - Je clique sur l'onglet "Virements", puis sur "Gestion des bénéficiaires".
  - S'affiche alors la liste de mes bénéficiaires.
  - Pour supprimer un bénéficiaire existant, je clique sur "Supprimer bénéficiaire".
  - Pour modifier le nom d'un bénéficiaire, je clique sur "Éditer bénéficiaire". Je peux alors saisir son nouveau nom et cliquer sur "Ok" pour valider ou sur "Annuler" si je change d'avis.
  - Pour ajouter un bénéficiaire, je clique sur "Nouveau bénéficiaire". Je peux saisir son non et son IBAN puis cliquer sur "Confirmer" pour valider l'ajout ou sur "Annuler" si je change d'avis.

5 Bilan

A l'issue de cette phase de conception/analyse/ébauche d'application bancaire, nous avons réalisé tout ce qui figurait dans notre cahier des charges initial, exception faite de la gestion de bases de données liées à notre interface graphique (ne figurait pas explicitement dans le cahier des charges, mais nous avions néanmoins cet objectif en tête).

Une des principales difficultés que nous avons rencontrées était liée à la liberté qui nous était sonnée sur le sujet. La grande question récurrente : jusqu'où doit aller notre application ? Nous avons donc procédé du plus global au plus précis, en commençant à modéliser des fonctionnalités de base d'une application bancaire, à savoir consulter le solde de ses comptes ou effectuer un virement unique par exemple.

Pour effectuer nos diagrammes de conception et d'analyse, nous avons utilisé le logiciel draw.io très adapté pour ce genre d'exercice.
Le rapport a quant à lui été réalisé sur le logiciel Overleaf qui permet une modification simultanée par plusieurs utilisateurs, l'utilisation de caractères spéciaux variés et offre une mise en page claire et agréable.

Au final, l'utilisation simultanée des logiciels Overleaf, draw.io, Éclipse ainsi que de GitLab Ensimag nous a permis de réaliser un travail de groupe efficace, malgré la distance et les situations de chacun.

6 Annexe : Diagrammes de Classe d'architecture MVC

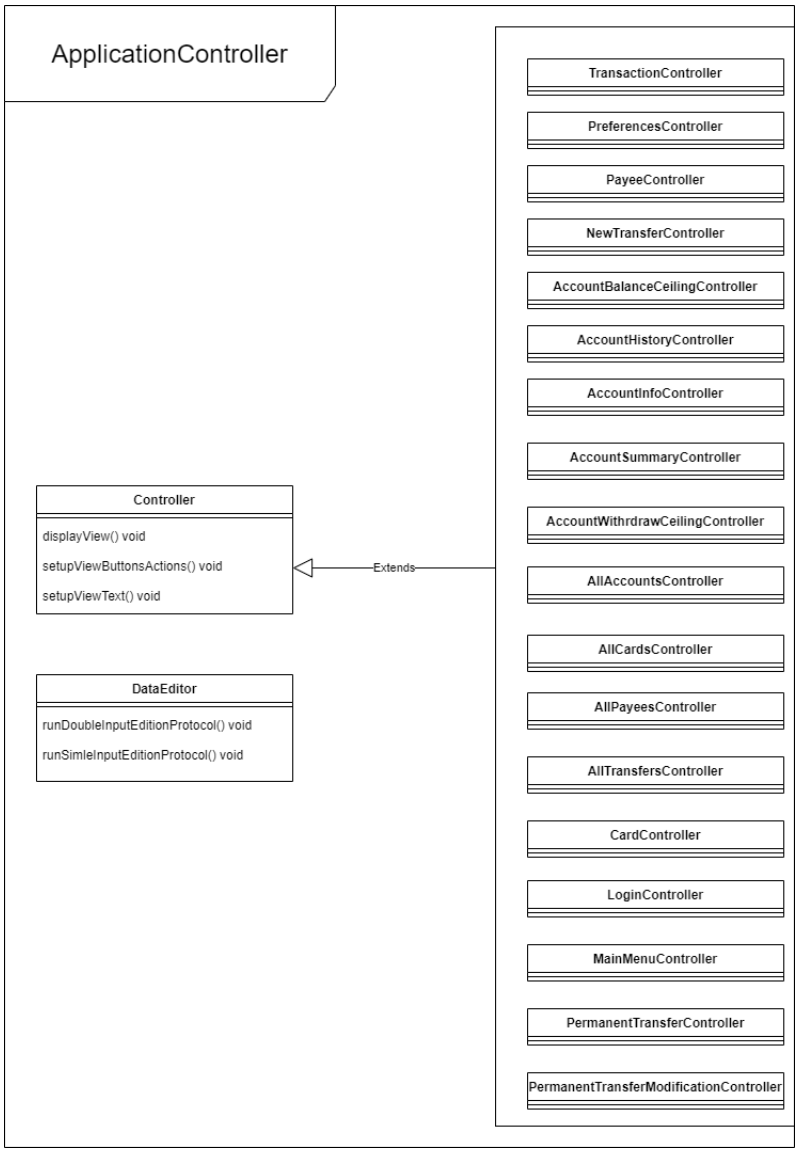
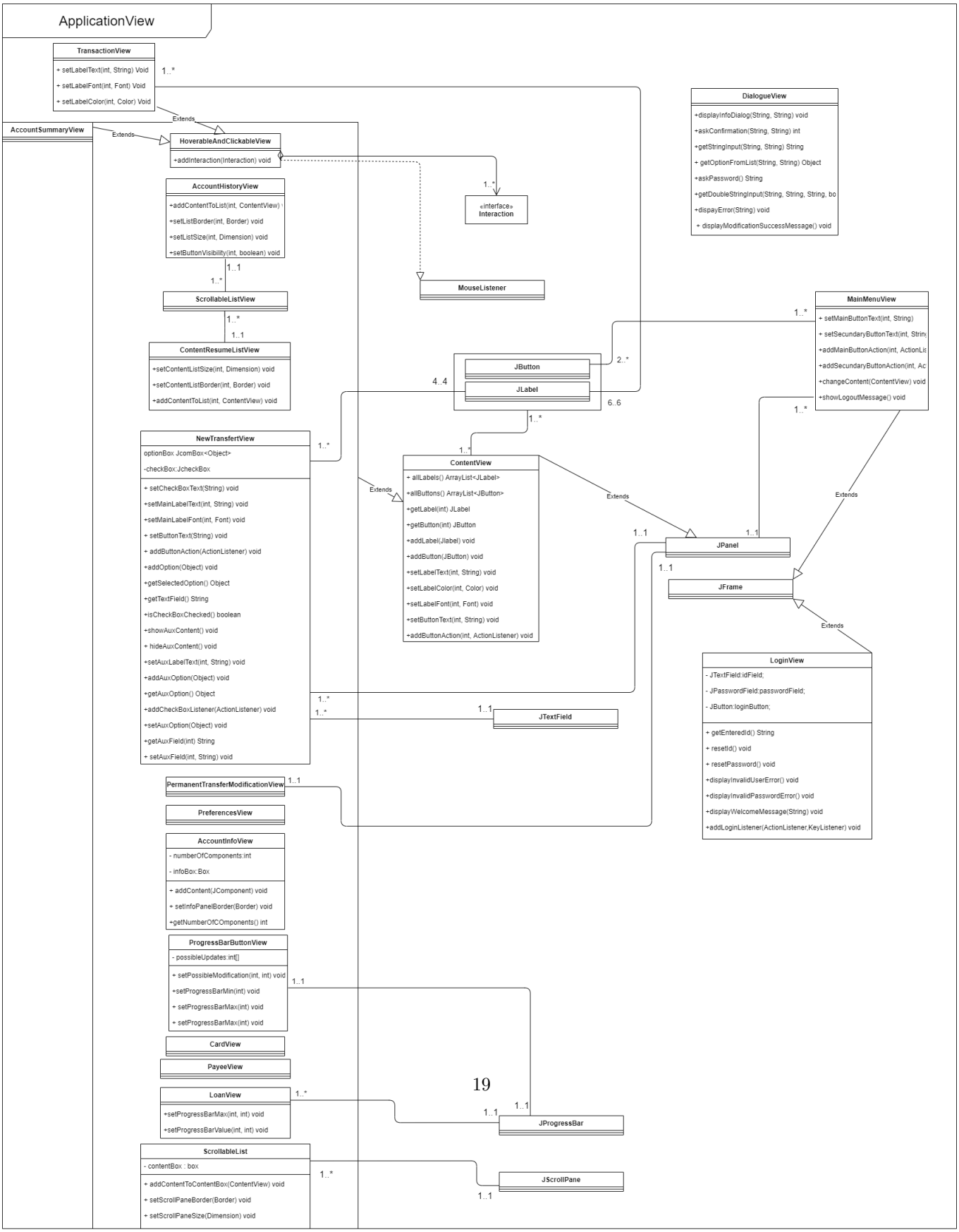
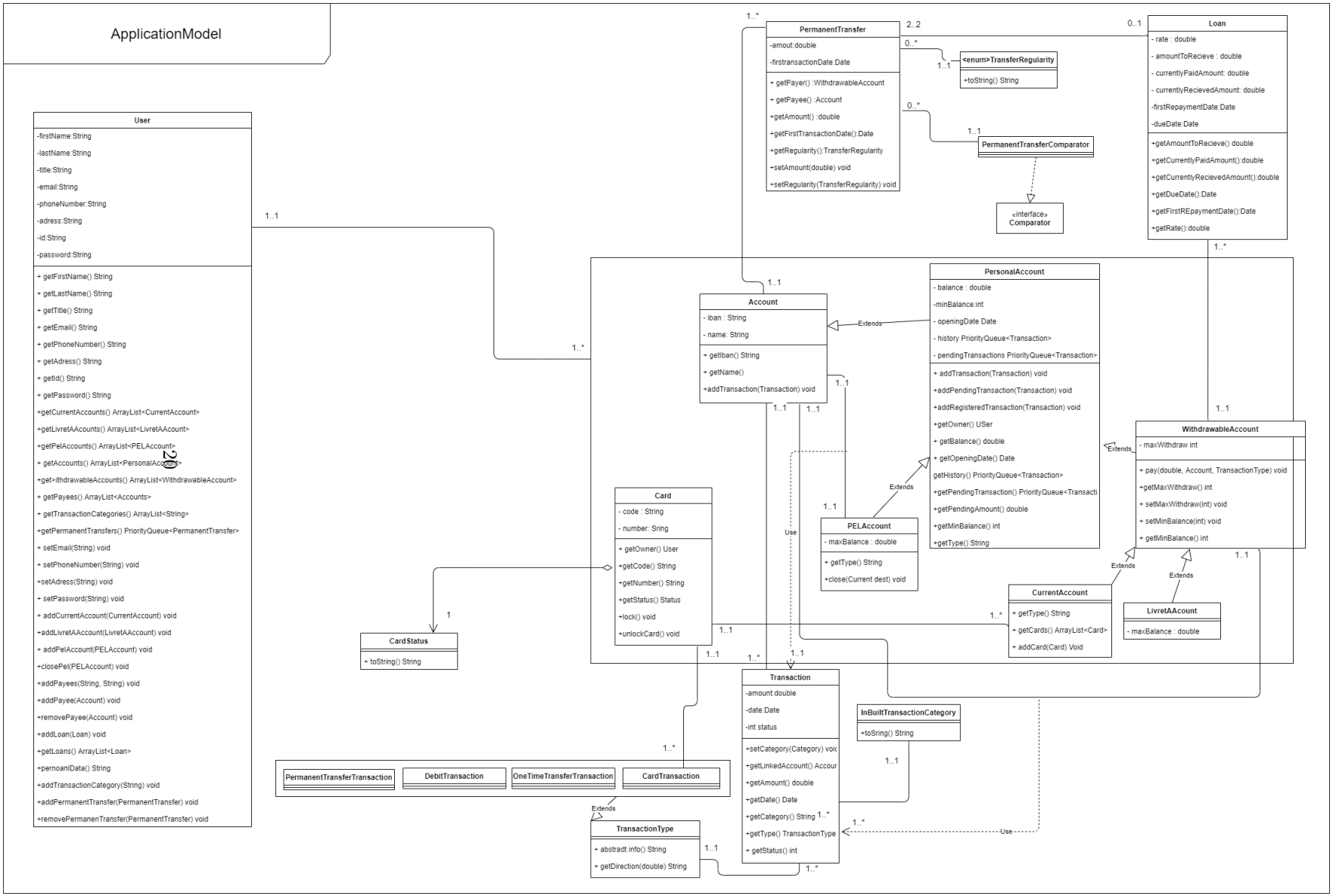


Figure 12: Diagramme de Conception partie Contrôler


ApplicationView
TransactionView
+ setLabelText(int, String) Void
+ setLabelFont(int, Font) Void
+ setLabelColor(int, Color) Void
1..*
Extends
AccountSummaryView
Extends
HoverableAndClickableView
+addInteraction(Interaction) void
1..*
«interface»
Interaction
MouseListener
DialogueView
+displayInfoDialog(String, String) void
+askConfirmation(String, String) int
+getStringInput(String, String) String
+ getOptionFromList(String, String) Object
+askPassword() String
+getDoubleStringInput(String, String, String, bo
+dispayError(String) void
+ displayModificationSuccessMessage() void
AccountHistoryView
+addContentToList(int, ContentView)
+setListBorder(int, Border) void
+setListSize(int, Dimension) void
+setButtonVisibility(int, boolean) void
1..1
1..*
ScrollableListView
1..*
1..1
ContentResumeListView
+setContentListSize(int, Dimension) void
+setContentListBorder(int, Border) void
+addContentToList(int, ContentView) void
JButton
JLabel
4..4
2..*
6..6
1..*
MainMenuView
+ setMainButtonText(int, String)
+ setSecundaryButtonText(int, Strin
+addMainButtonAction(int, ActionLis
+addSecundaryButtonAction(int, Ac
+changeContent(ContentView) void
+showLogoutMessage() void
1..*
NewTransfertView
optionBox JcomBox<Object>
-checkBox:JcheckBox
+ setCheckBoxText(String) void
+setMainLabelText(int, String) void
+setMainLabelFont(int, Font) void
+ setButtonText(String) void
+ addButtonAction(ActionListener) void
+addOption(Object) void
+getSelectedOption() Object
+getTextField() String
+isCheckBoxChecked() boolean
+showAuxContent() void
+ hideAuxContent() void
+setAuxLabelText(int, String) void
+addAuxOption(Object) void
+getAuxOption() Object
+addCheckBoxListener(ActionListener) void
+setAuxOption(Object) void
+getAuxField(int) String
+ setAuxField(int, String) void
1..*
Extends
ContentView
+ allLabels() ArrayList<JLabel>
+allButtons() ArrayList<JButton>
+getLabel(int) JLabel
+getButton(int) JButton
+addLabel(Jlabel) void
+addButton(JButton) void
+setLabelText(int, String) void
+setLabelColor(int, Color) void
+setLabelFont(int, Font) void
+setButtonText(int, String) void
+addButtonAction(int, ActionListener) void
Extends
1..1
1..1
JPanel
1..1
JFrame
Extends
Extends
LoginView
- JTextField:idField;
- JPasswordField:passwordField;
- JButton:loginButton;
+ getEnteredId() String
+ resetId() void
+ resetPassword() void
+displayInvalidUserError() void
+displayInvalidPasswordError() void
+displayWelcomeMessage(String) void
+addLoginListener(ActionListener,KeyListener) void
1..*
1..*
1..1
JTextField
PermanentTransferModificationView
1..1
PreferencesView
AccountInfoView
- numberOfComponents:int
- infoBox:Box
+ addContent(JComponent) void
+ setInfoPanelBorder(Border) void
+getNumberOfComponents() int
ProgressBarButtonView
- possibleUpdates:int[]
+ setPossibleModification(int, int) void
+setProgressBarMin(int) void
+ setProgressBarMax(int) void
+ setProgressBarMax(int) void
1..1
CardView
PayeeView
LoanView
+setProgressBarMax(int, int) void
+setProgressBarValue(int, int) void
1..*
1..1
1..1
JProgressBar
ScrollableList
- contentBox : box
+ addContentToContentBox(ContentView) void
+ setScrollPaneBorder(Border) void
+ setScrollPaneSize(Dimension) void
1..*
1..1
JScrollPane


ApplicationModel
User
-firstName:String
-lastName:String
-title:String
-email:String
-phoneNumber:String
-adress:String
-id:String
-password:String
+ getFirstName() String
+ getLastName() String
+ getTitle() String
+ getEmail() String
+ getPhoneNumber() String
+ getAdress() String
+ getId() String
+ getPassword() String
+getCurrentAccounts() ArrayList<CurrentAccount>
+getLivretAAcounts() ArrayList<LivretAAcount>
+getPelAccounts() ArrayList<PELAccount>
+ getAccounts() ArrayList<PersonalAccount>
+get>ithdrawableAccounts() ArrayList<WithdrawableAccount>
+ getPayees() ArrayList<Accounts>
+ getTransactionCategories() ArrayList<String>
+getPermanentTransfers() PriorityQueue<PermanentTransfer>
+ setEmail(String) void
+ setPhoneNumber(String) void
+setAdress(String) void
+ setPassword(String) void
+ addCurrentAccount(CurrentAccount) void
+addLivretAAccount(LivretAAccount) void
+ addPelAccount(PELAccount) void
+closePel(PELAccount) void
+addPayees(String, String) void
+addPayee(Account) void
+removePayee(Account) void
+addLoan(Loan) void
+getLoans() ArrayList<Loan>
+pernoanlData() String
+addTransactionCategory(String) void
+addPermanentTransfer(PermanentTransfer) void
+removePermanenTransfer(PermanentTransfer) void
PermanentTransfer
-amout:double
-firstransactionDate:Date
+ getPayer() :WithdrawableAccount
+ getPayee() :Account
+getAmount() :double
+getFirstTransactionDate():Date
+getRegularity():TransferRegularity
+setAmount(double) void
+setRegularity(TransferRegularity) void
<enum>TransferRegularity
+toString() String
PermanentTransferComparator
«interface»
Comparator
Loan
- rate : double
- amountToRecieve : double
- currentlyPaidAmount: double
- currentlyRecievedAmount: double
-firstRepaymentDate:Date
-dueDate:Date
+getAmountToRecieve() double
+getCurrentlyPaidAmount():double
+getCurrentlyRecievedAmount():double
+getDueDate():Date
+getFIrstREpaymentDate():Date
+getRate():double
Account
- iban : String
- name: String
+ getIban() String
+ getName()
+addTransaction(Transaction) void
PersonalAccount
- balance : double
-minBalance:int
- openingDate Date
- history PriorityQueue<Transaction>
- pendingTransactions PriorityQueue<Transaction>
+ addTransaction(Transaction) void
+addPendingTransaction(Transaction) void
+addRegisteredTransaction(Transaction) void
+getOwner() USer
+ getBalance() double
+ getOpeningDate() Date
getHistory() PriorityQueue<Transaction>
+getPendingTransaction() PriorityQueue<Transacti
+getPendingAmount() double
+getMinBalance() int
+getType() String
WithdrawableAccount
- maxWithdraw int
+ pay(double, Account, TransactionType) void
+getMaxWithdraw() int
+ setMaxWithdraw(int) void
+ setMinBalance(int) void
+ getMinBalance() int
Card
- code : String
- number: Sring
+ getOwner() User
+getCode() String
+getNumber() String
+getStatus() Status
+lock() void
+unlockCard() void
PELAccount
- maxBalance : double
+ getType() String
+close(Current dest) void
CurrentAccount
+ getType() String
+ getCards() ArrayList<Card>
+ addCard(Card) Void
LivretAAcount
- maxBalance : double
CardStatus
+ toString() String
Transaction
-amount:double
-date:Date
-int status
+setCategory(Category) void
+getLinkedAccount() Accour
+getAmount() double
+getDate() Date
+getCategory() String
+getType() TransactionType
+ getStatus() int
InBuiltTransactionCategory
+toSring() String
PermanentTransferTransaction
DebitTransaction
OneTimeTransferTransaction
CardTransaction
TransactionType
+ abstradt info() String
+ getDirection(double) String
Extends
Use
1..1
1..*
0..*
2..2
0..1
1