

Projet de programmation et conception orientées objet



Trading Simulator



Ricardo Corona / GI02

Raphaël Nachar / GI01

Jérôme Raynal / GI02

Jonathan Rojas / GI02

Amira Stoleru / GI01

Sommaire

Introduction	3
Description du contexte	4
Description de l'architecture	6
Possibilité d'évolution de l'architecture	8
Planning et organisation en groupe du projet	13
Conclusion	14
Annexes	15

Introduction

Dans le cadre de l'UV LO21, nous avons réalisé une application Trading Simulator. Il s'agit d'un programme de simulation de stratégies de trading codé en C++ avec le framework Qt pour l'interface graphique.

Ce rapport a pour but de vous présenter le contexte et le but de ce projet, ainsi que de décrire et justifier l'architecture de notre application et de ses éventuelles évolutions puis enfin de vous présenter nos méthodes de travail et comment nous nous sommes organisés pour réaliser et mener à bout ce projet.

Au sein de ce dossier, vous trouverez également une documentation en html générée avec Doxygen, le code source du projet et une courte vidéo de présentation de l'application qui démontre le bon fonctionnement de chaque fonctionnalité attendue.

I. Description du contexte

Nous avons développé une application afin de pratiquer la conception et la programmation orientée objet en C++. Notre sujet a porté sur la réalisation d'un programme de simulation de stratégies de trading.

Les technologies utilisées ont été les suivantes: framework Qt (Qt Creator, IDE de Qt et un assistant de création d'interface graphique, Qt Designer) ainsi que le logiciel de génération de documentation, Doxygen.

Nous nous sommes basés sur le polycopié de cours et la documentation Qt. Nos livrables sont le code source, la documentation Doxygen et une vidéo de présentation de l'application.

Notre application est capable de recevoir des cours OHLCV (Open, High, Low, Close et Volume) par deux voies: à partir d'un fichier csv ou manuellement. Dans le menu principal, on peut créer de nouvelles devises (code, nom de la devise et zone géographique), ajouter de nouvelles paires de devises à partir de deux devises différentes et ajouter les cours OHLCV manuellement pour une paire donnée. On peut également voir un tableau récapitulatif des données sur le cours d'une paire et afficher le graphique en bougie associé. Finalement, on peut sélectionner et paramétrer trois modes de simulation différents: le mode manuel, le mode pas à pas et le mode automatique.

Pour toutes les simulations, l'utilisateur commence par choisir une paire de devises, la date de début de la simulation, le montant de la devise de base/contrepartie ainsi que le taux de commission du broker. Il dispose toujours d'un éditeur de texte et de l'historique de ses transactions.

Voici comment nous avons compris les différentes fonctionnalités des modes de simulations:

Mode manuel : On peut ajouter transactions à n'importe quelle date (futur, passé, présent). L'utilisateur choisi les dates d'achat et de vente pour la paire chargée et à la possibilité d'annuler la dernière transaction ajoutée.

Mode pas à pas (type de mode manuel plus réaliste): On ne connaît que le prix d'ouverture du jour saisi par l'utilisateur. On peut faire des transactions lors du jour en

cours, ou dans le futur mais toujours au prix d'ouverture du jour courant. On peut revenir en arrière mais les transactions sont effacés. Ce mode est censé être plus réaliste car on ne peut réaliser les transactions qu'avec le prix d'ouverture du jour courant qui est le seul connu. De plus, il n'y a pas d'affichage de bougie dans cette partie car cela n'a pas de sens. En effet, une bougie nécessite la connaissance des informations complètes sur le cours OHLC, or ici on n'a que le prix d'ouverture.

Mode automatique : L'utilisateur a la possibilité de sélectionner une stratégie de trading parmi une liste de stratégies proposées. Pour les stratégies de trading, on utilise les indicateurs techniques suivants: RSI (Relative Strength Index), MACD (Moving Average Convergence Divergence) et EMA (Exponential Moving Average).

II. Description de l'architecture

Dans un premier temps, avant toute réalisation, nous avons décidé d'établir un modèle conceptuel UML de notre projet afin d'avoir une idée plus claire de ce que nous devons réaliser. Nous avons donc réalisé un premier jet d'UML (que vous pouvez retrouver en *Annexe 1*) permettant de définir une première base de l'architecture du programme par des relations entre différentes classes. Cette première version a bien évolué au cours du projet jusqu'à notre UML final (en *Annexe 2*), notamment au fur et à mesure que nous avons compris de mieux en mieux les tâches à réaliser.

Tout d'abord, le projet est caractérisé par l'utilisation et la manipulation de devises. Nous avons donc choisi de créer une classe *Devise* contenant en attributs toutes les informations qui lui sont liées (code, monnaie, zone). Pour plus de sûreté, nous avons décidé de mettre en place le principe d'encapsulation pour toutes nos classes, c'est à dire de mettre les attributs de nos classes en privée pour ne pas qu'elle puisse être directement modifiés par un utilisateur mais seulement accessibles par des méthodes tel que *getCode()*.

Ensuite, nous devons créer des paires de devises, nous avons donc créé une classe *PaireDevises* qui lie deux devises, une devise de *base* et une devise de *contrepartie* avec un *surnom*.

A partir de là, nous devons d'une part, créer des cours OHLC pour une paire de devises. De ce fait, nous avons ajouté une classe *OHLC* avec tous les attributs nécessaires (comme la date, le volume et les cours OHLC). On y a par la suite ajouté en attributs les différents indicateurs pour savoir, pour un cours, à quel niveau de l'indicateur nous sommes.

De ces cours OHLC, nous devons les représenter sous forme de bougie. Pour réaliser les bougies, nous avons créé une classe *Bougie* qui a pour attribut la paire associée, ainsi que la mèche inférieure et supérieure, la taille de son corps, sa tendance et son type si elle en a un (toupie, doji...) pour permettre notamment de réaliser les indicateurs que nous aborderons juste après.

Les cours OHLC d'une paire de devises évoluent constamment tout au long du temps. C'est pourquoi, nous devons enregistrer cette évolution avec la classe *EvolutionCours* qui enregistre les différents cours d'une paire de devises. Cette classe a donc pour attribut le nombre de cours enregistrés pour une paire de devise. Nous avons aussi implémenté un design pattern iterator afin de pouvoir parcourir et afficher tous les cours OHLC sur une plage (date) donnée. Cette iterator est aussi utile pour former les

bougies et générer les graphiques de bougie OHLC. Enfin, nous avons ajouté des attributs afin de gérer les différents indicateurs.

Comme nous l'avons effectué en TD, nous avons opté pour une classe de gestion des devises nommé *DevisesManager* qui a pour rôle de gérer les devises et notamment de récupérer les informations via ses méthodes des devises impliqués dans le programme. Elle a pour attributs un tableau de devises, ainsi que le nombre de devises (et le nombre maximum de devises afin d'agrandir le tableau lorsque cela est nécessaire) et de même pour les paires de devises (tableau, nombre et nombre maximum). Enfin cette classe a aussi un tableau d'objets de type *EvolutionCours* (ainsi que nombre et nombre maximum).

Dans la suite du projet, nous devons réaliser des simulations automatiques et manuelles. Dans la première version de notre UML, nous avons pensé à réaliser une classe *simulation* générale avec des classes héritant de cette dernière en fonction du mode automatique ou manuel avec des classes stratégies différentes. Nous pensions aussi réaliser une classe transaction avec une classe les gérant transactionManager.

Cependant, lors de l'implémentation, nous avons préféré réaliser une seule classe *simulation* comprenant des méthodes permettant d'effectuer des transactions (achat et revente) mais aussi les fonctions de stratégies qui consistent à définir des critères de stratégies à partir duquel on achète et on vend.

Enfin pour définir ces stratégies, nous devons ajouter des indicateurs sur les cours. Pour gérer cela, nous avons créer une classe indicateurs qui gèrent les indicateurs EMA,RSI et MACD.

Pendant l'implémentation, nous avons aussi créer des classes afin de réaliser les différentes interface Qt. Nous avons préféré ne pas les représenter sur l'UML pour plus de compréhension de la conception de notre architecture.

III. Possibilité d'évolution de l'architecture

Nous avons organisé l'interface de TradingSimulator afin que l'utilisateur puisse facilement la piloter. Les fenêtres sont liées et l'on peut facilement revenir en arrière ou quitter l'application.

En effet, comme on peut le voir sur la *figure 1*, la fenêtre d'ouverture permet de choisir le mode de saisie des données.

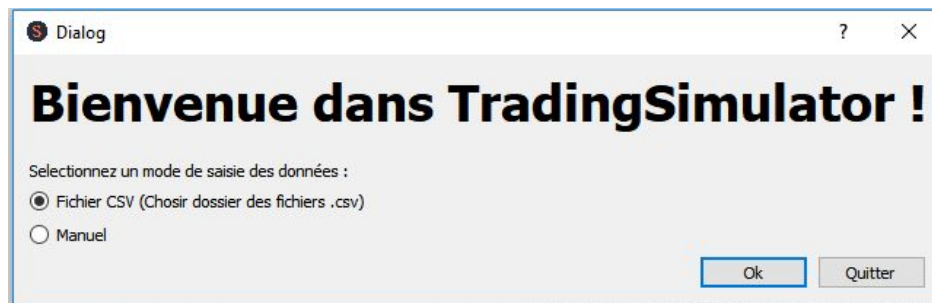


Figure 1: Fenêtre d'ouverture de l'application

Ensuite, après avoir choisi un des boutons radio et "Ok", l'utilisateur arrive au menu principal souhaité où il peut choisir parmi plusieurs options.

Pour le mode de saisie de données manuel :

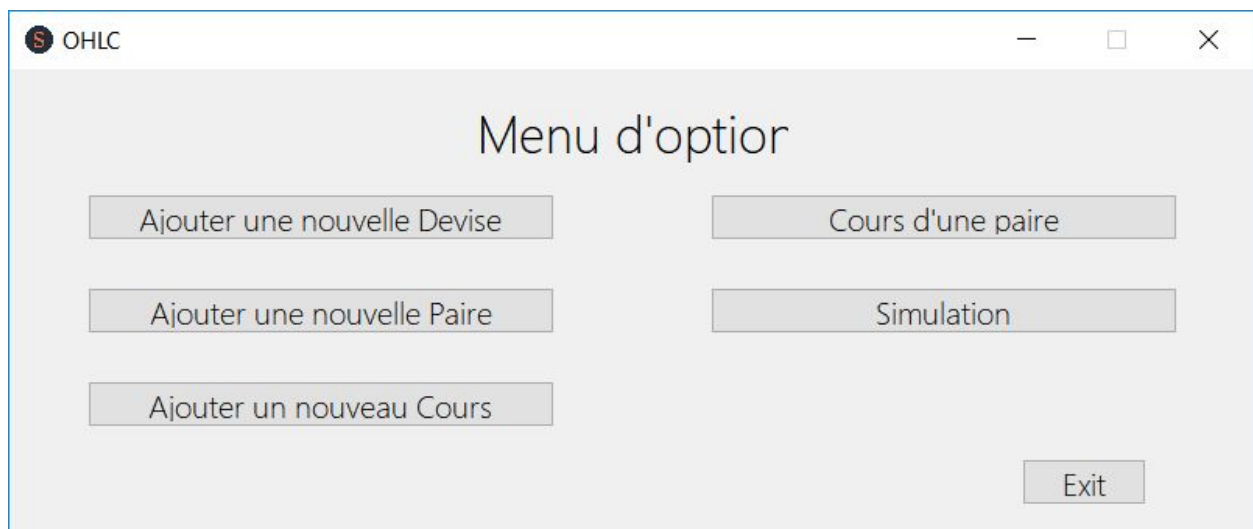
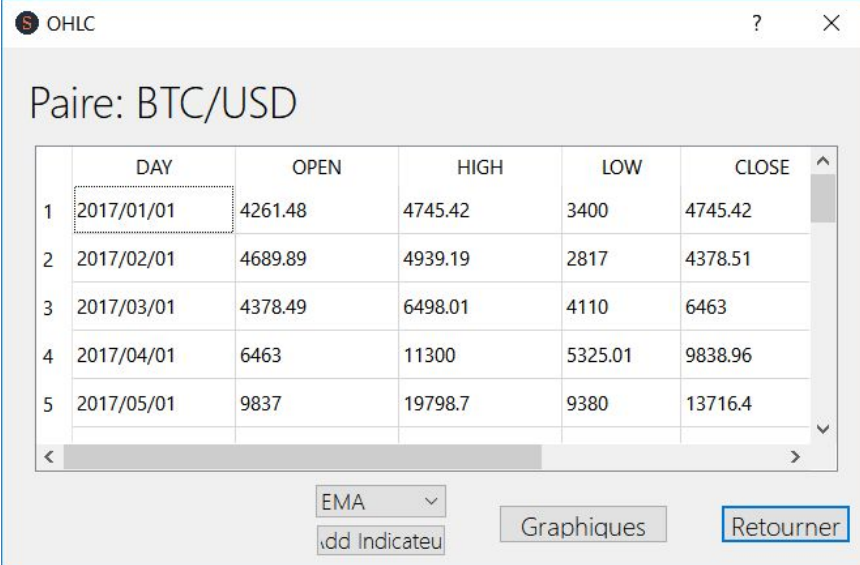


Figure 2: Menu principal du mode manuel

Dans ce mode, nous pouvons ajouter une Devise, une paire de devises et des cours OHLC.

Si les informations nécessaires sont disponibles, en cliquant sur “Cours d’une paire”, la fenêtre de la *figure 3* s’affiche. Celle-ci nous permet de lire le cours OHLCV pour un jour donné.



Paire: BTC/USD

	DAY	OPEN	HIGH	LOW	CLOSE
1	2017/01/01	4261.48	4745.42	3400	4745.42
2	2017/02/01	4689.89	4939.19	2817	4378.51
3	2017/03/01	4378.49	6498.01	4110	6463
4	2017/04/01	6463	11300	5325.01	9838.96
5	2017/05/01	9837	19798.7	9380	13716.4

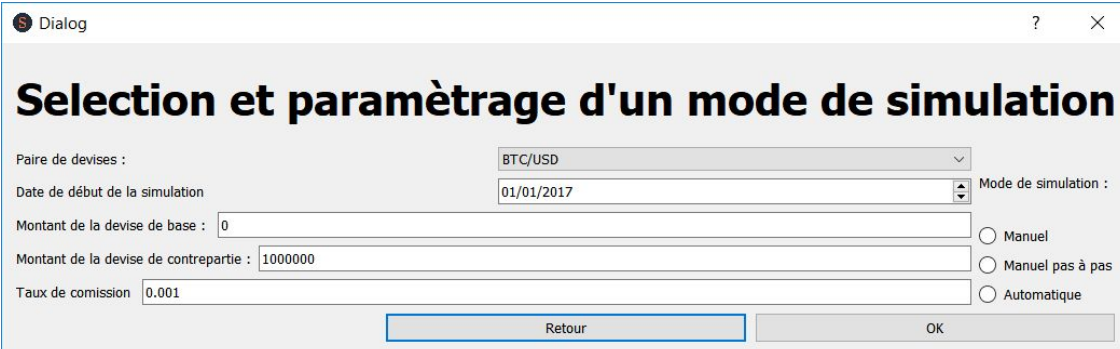
EMA
Add Indicateur
Graphiques
Retourner

Figure 3: Cours OHLCV d’une paire de devise

Lorsque l’utilisateur sélectionne le bouton “Graphiques” de la *figure 3*, le graphique en bougie de la paire s’affiche ainsi que le graphique avec les différents indicateurs sélectionnés.

Figure 4: Affichage du graphique

En revenant au menu principal et choisissant l’option “Simulation”, la fenêtre de la *figure 5* s’ouvre et permet de paramétrer un type de simulation.



Selection et paramétrage d'un mode de simulation

Paire de devises : BTC/USD

Date de début de la simulation : 01/01/2017

Montant de la devise de base : 0

Montant de la devise de contrepartie : 1000000

Taux de comission : 0.001

Mode de simulation :

☐ Manuel

☐ Manuel pas à pas

☐ Automatique

Retour OK

Figure 5: Choix d’un mode de simulation

Enfin, voici l'interface des trois modes de simulation: simulation manuelle classique (*figure 6*), simulation manuelle pas à pas (*figure 7*) et simulation automatique (*figure 8*).

Figure 6: interface de la simulation manuelle classique

Figure 7: interface de la simulation manuelle pas à pas

Figure 8: interface de la simulation automatique

Afin de rendre l'application évolutive d'un point de vue architectural, nous avons essayé de faciliter au maximum l'implémentation de nouvelles fonctionnalités telles que la modification du format des données d'entrée et l'ajout d'indicateurs et de stratégies de trading.

- **Modification du format des données fournies en entrée :**

Actuellement, le format d'entrée est le format csv (comma-separated values). Ceci signifie que toutes les informations distinctes présentes dans le fichier sont séparées par des virgules. Nous avons spécifié ceci dans le code, donc si jamais les données du nouveau type de fichier d'entrée étaient maintenant séparées par un autre caractère (comme "/" ou "."), il suffirait de remplacer la virgule par le nouveau caractère dans le code.

Si l'utilisateur souhaite avoir la possibilité de rentrer plusieurs types de fichiers, on peut envisager une option où l'on demande par quel caractère sont séparées les données et ajuster par la suite le programme en conséquence.

- Ajout d'indicateurs et de stratégies automatiques de trading:

Depuis le menu principal, on peut voir les données sur le cours d'une paire de devises. Il est possible ici d'ajouter des indicateurs sur les graphiques, que l'on peut ensuite visualiser dans "Graphiques".

Nous avons rajouté un QComboBox avec les types de indicateurs dans la fenêtre où l'on peut voir le tableau avec les cours d'une paire.

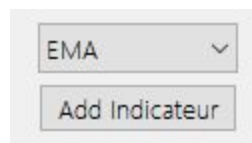


Figure 9: Widgets pour ajouter un indicateur

Après avoir cliqué sur *Add Indicateur* et dépendamment du type choisi, une autre interface s'affiche.

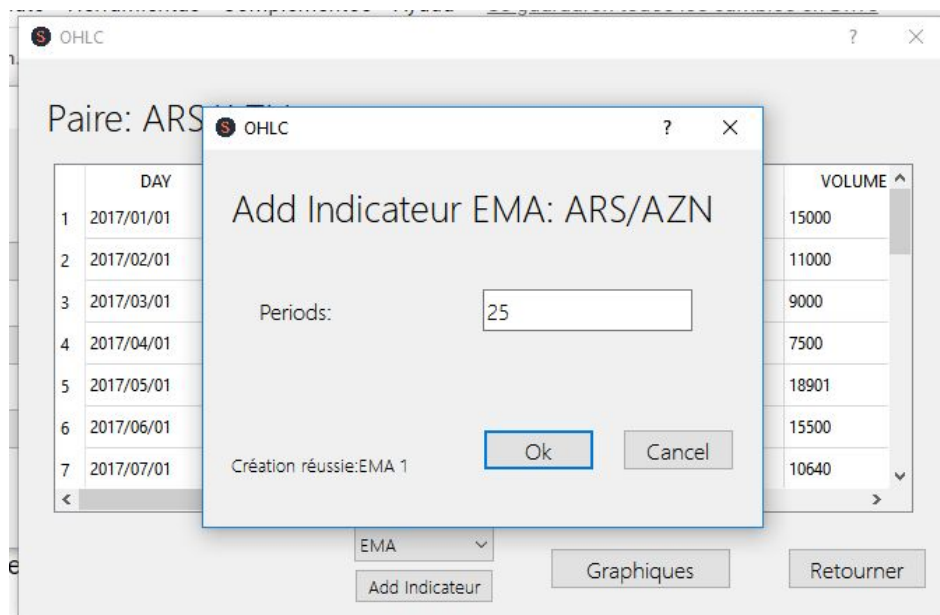


Figure 10: Interface pour ajouter un indicateur ex. EMA

Maintenant pour voir visualiser l'indicateur ajouté on va dans la partie graphique où il existe l'option de sélectionner un indicateur ajouté par l'utilisateur et un bouton pour le faire apparaître.



Figure 11: Interface des graphiques avec indicateur ajouté

Si l'on veut rajouter de nouveaux indicateurs, il suffit d'adapter la classe *Indicateurs*. Et ensuite, l'incorporer dans une stratégie de trading de la simulation automatique.

IV. Planning et organisation en groupe du projet

Pour commencer, nous nous sommes tous réunis afin de discuter et comprendre ensemble le sujet du projet. Ceci nous a permis de s'assurer que nous avons tous bien saisis les enjeux du sujet et d'éclaircir certains points équivoques.

Ensuite, nous avons repris et réajuster les éléments que nous avons commencé lors des TD sur cette partie. Nous avons également essayé de comprendre comment fonctionnait Qt et avons choisi d'utiliser Qt Designer pour réaliser l'interface graphique de l'application.

La première chose que nous avons fait a été de réaliser une note de clarification

contenant la première ébauche de l'UML. Nous l'avons complété au fur et à mesure du projet et elle nous a permis de s'y référer afin d'éclairer les points plus compliqués mais aussi de ne pas oublier toutes les fonctions que l'application devaient remplir.

Nous avons également créé un projet sur Gitlab afin que l'on puisse travailler de façon organisée ensemble et réunir tous nos morceaux de code. Pour réaliser et avancer progressivement tout au long du semestre, nous avons décidé de nous réunir une fois toutes les deux semaines minimum pour faire un point sur les avancements de chacun. Entre chaque réunion, chacun avait une tâche à réaliser tout seul ou avec un autre membre de l'équipe.

Enfin, nos réunions nous ont permis de voir l'avancement global du projet et de résoudre ensemble certains problèmes ou incompréhension du sujet. Enfin, nous nous sommes répartis les tâches afin de générer la documentation Doxygen, faire la vidéo de présentation et le rapport final.

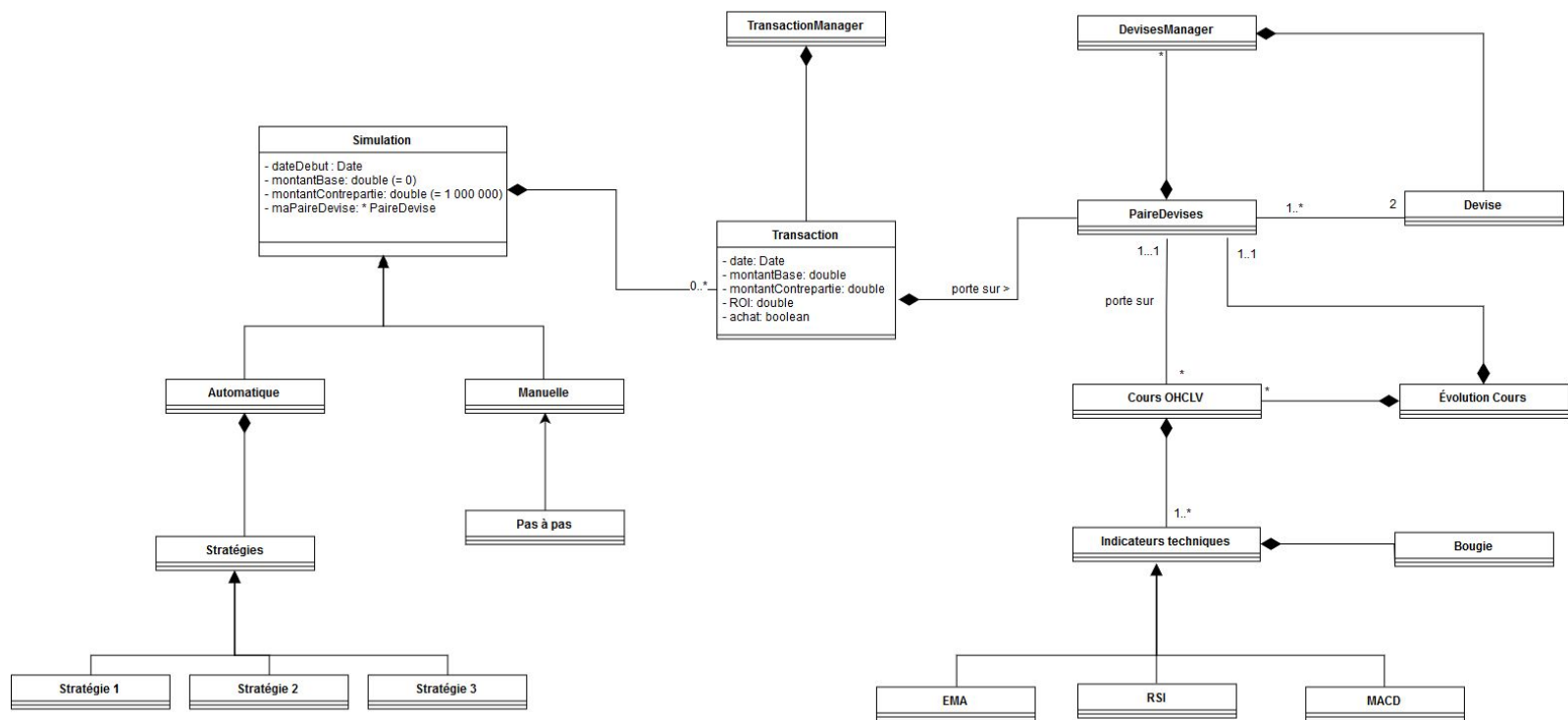
Conclusion

Pour conclure, la réalisation de ce Trading Simulator nous a permis d'appliquer les notions apprises sur la conception et la programmation orientée objet. Nous avons également appris à travailler en groupe sur un projet de plus grande envergure à l'aide de Gitlab notamment. C'était l'occasion de mettre en application les concepts vus tout au long du semestre.

De manière générale, nous avons appris à utiliser les concepts de programmation orientée objet afin de répondre à un problème donné.

Annexes

Annexe 1: Premier jet du modele conceptuel UML



Annexe 2: UML final de notre application

