RAPPORT TECHNIQUE

PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

DANS LE CADRE DU COURS LOG795 - PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

ANALYSE DE SENTIMENTS EN FINANCE

BENJAMIN LEBOIS (LEBB12099302)

DAVID MÉTHOT (METD25129108)

SÉBASTIEN SANTERRE (SANS12109103)

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeurs-superviseurs

ALAIN APRIL

PATRICK CARDINAL

MONTRÉAL, 13 Avril 2017

HIVER 2017

REMERCIEMENTS

La réalisation de ce projet n'a pu être possible sans la contribution de personnes clés; nous tenons à les remercier. On tient à remercier le département de Génie Logiciel et des TI pour l'opportunité de travailler sur un projet qui se rapproche du marché du travail.

Tout d'abord, notre équipe de projet tient à remercier les professeurs Alain April et Patrick Cardinal, du département de Génie Logiciel à l'École de Technologie Supérieure, pour leurs efforts continus tout au long du projet. Ils ont su assurer un suivi assidu hebdomadaire nous permettant de progresser d'une manière efficace et constante.

Ensuite, nous tenons à remercier Thomas Maketa, étudiant au doctorat en Génie Logiciel à l'École de Technologie Supérieure, pour l'opportunité de projet et pour son transfert de connaissances du domaine de la finance. Ces notions ont été cruciales pour la réalisation de ce projet.

Finalement, nous soulignons le travail effectué précédemment par Nicolas Hubert, étudiant au baccalauréat en Génie Logiciel à l'École de Technologie Supérieure. Son travail est la base du présent projet et rapport technique.

RÉSUMÉ

PROJET D'ANALYSE DES SENTIMENTS EN FINANCE

Ce rapport technique présente le travail réalisé pour une plateforme web de backtesting sur la gestion d'actions et d'options boursières. Ce travail reprend le précédent ouvrage laissé par l'étudiant Nicolas Hubert. L'ouvrage contient, entre autres, une plateforme qui permet à un internaute d'écrire des stratégies boursières en Python, de les exécuter et de consulter le rendement de ses actifs. La plateforme se nomme PyAlgoTrade et est open source. La solution développée dans le cadre de ce projet final comporte l'ajout d'une interface utilisateur permettant d'utiliser les fonctionnalités de la plateforme et d'une structure de code pour utiliser les options boursières. Le travail a pour but de supporter l'exécution de stratégies de base en finances telles que: Covered call, Bear Put Spread, Butterfly et Strangle. Chaque module développé est expliqué et justifié. L'implémentation de la solution respecte les concepts de programmation orientée objet, soit: l'encapsulation, l'héritage et le polymorphisme. De plus, des recommandations sont fournies à la toute fin du document pour orienter les travaux futurs.

TABLE DES MATIÈRES

REMERCIEMENTS **1**

RÉSUMÉ **2**

PROJET D'ANALYSE DES SENTIMENTS EN FINANCE 2

TABLE DES MATIÈRES **3**

LISTE DES FIGURES **5**

GLOSSAIRE **6**

INTRODUCTION **8**

CHAPITRE 1 - DÉTERMINATION DE L'OBJECTIF DU PROJET **9**

1.1 Contexte et problématique 9

1.2 Objectifs 9

CHAPITRE 2 - APPROCHE **10**

2.1 Méthodologie 10

2.2 Techniques et outils 10

2.2.1 Gestion de code source 10

2.2.2 Gestion de projet 10

2.2.3 Outils de développement 10

2.2.4 Outils de déploiement 10

2.3 Livrables et planifications 11

2.4 Risques 12

CHAPITRE 3 - PYALGOTRADE **14**

3.1 Introduction au système PyAlgoTrade 14

3.1.1 Pourquoi PyAlgoTrade? 14

CHAPITRE 4 - RÉUSINAGE **15**

4.1 Motivations 15

4.2 Stratégie 15

4.3 Modifications 15

CHAPITRE 5 - NOTRE PLATEFORME **17**

5.1 Besoins 17

5.2 Inspirations 17

5.3 Choix technologiques 17

5.4 Exécution de code source 18

5.5 Source de données 19

5.6 Déploiement 19

5.7 Rédaction de stratégies boursières 20

CHAPITRE 6 - TESTS **21**

6.1 Objectif des tests 21
6.2 Stratégie de test 21
6.3 Implémentation des tests et bogues trouvés 21
6.4 Résultat des tests 22
6.5 Changements dans le futur 23

CHAPITRE 7 - EFFORTS INDIVIDUELS 24
7.1 Répartition de la charge de travail 24

CHAPITRE 8 - FUTUR DU PROJET 25
8.1 Limitations 25
8.1.1 Opérations sur les options 25
8.1.2 Données avec plusieurs strike price et expiry date 25
8.1.3 Source de données 26
8.1.4 Interface 26
8.2 Recommandations 26
8.2.1 Opérations sur les options 26
8.2.2 Données avec plusieurs strike price et expiry date 26
8.2.3 Source de données 26
8.2.4 Interface 27
8.3 Exigences pour supporter les options 27

CONCLUSION 29

RÉFÉRENCES 30

BIBLIOGRAPHIE 31

ANNEXE I - Guide de déploiement 32

ANNEXE II - Compte-rendu de réunions 36

LISTE DES FIGURES

Nom de la figure	Page
Figure 1 - WBS	11
Figure 2 - Diagramme de classes de Broker	15
Figure 3 - Diagramme de classes de Strategy	15
Figure 4 - Dossier de données	18
Figure 5 - Sélection de la source de données	18
Figure 6 - Diagramme de déploiement	19
Figure 7 - Arborescence de testcases	21

GLOSSAIRE

Certains acronymes, sigles et abréviations utilisés dans ce document peuvent porter à confusion. Ceux-ci proviennent du jargon de la finance et du domaine de l'informatique. Ces termes sont définis dans le tableau ci-dessous afin d'aider le lecteur à comprendre le rapport technique ci-présent.

Terme	Définition
Stock / Action	Titre de propriété délivré par une société de capitaux. Elle permet de détenir une partie du capital d'une entreprise (ex.: Google).
Stock-option / Option	Contrat de possibilité d'achat ou de vente d'une action entre un acheteur et un vendeur à une date et à un prix fixe.
Instrument / Titre	Société de capitaux qui possède des actions.
CALL	Option d'achat d'une action pour un titre à un prix et une date fixe d'une société de capital envers un vendeur.
PUT	Option de vente d'une action pour un titre à un prix et une date fixe d'une société de capital envers un vendeur.
Expiry Date (Date d'expiration)	Date limite pour exercer une option. Après cette date l'option cesse d'exister
Strike Price	Prix fixé auquel le détenteur d'une option peut acheter (CALL) ou vendre (PUT) le titre en question. [1]
Broker	Courtier d'interaction avec notre système et le système de la bourse ou dans le monde des finances, le logiciel qui est utilisé pour acheter/vendre des actions
Buy	Achat d'une action boursière.
Sell	Vente d'une action boursière.
Python	Langage de programmation.
PyAlgoTrade	Plateforme de backtesting et d'échanges boursiers.
Backtesting	Technique qui sert à tester la pertinence d'une stratégie financière en s'appuyant sur un ensemble de données historiques réelles.
Feed	Source d'informations. Dans notre cas, on parle d'articles provenant du réseau social Twitter.
Fichier CSV	Fichier avec des lignes de données séparées par des virgules.

Work Breakdown Structure (WBS)	Organigramme des tâches du projet ou encore structure de découpage du projet, est une décomposition hiérarchique des travaux nécessaires pour réaliser les objectifs du projet.
Open-source	Licence qui permet à des développeurs de modifier et de redistribuer du code source pour des travaux dérivés.
Option in-the-money	L'option d'achat est dite in-the-money si sont strike price est inférieur au prix du titre. L'option de vente est dite in-the-money si sont strike price est supérieur au prix du titre. [1]
Option out-of-the-money	L'option d'achat est dite out-of-the-money si sont strike price est supérieur au prix du titre. L'option de vente est dite out-of-the-money si sont strike price est inférieur au prix du titre. [1]

INTRODUCTION

Le marché de la finance a bien évolué avec le temps. Autrefois, les transactions s'effectuaient par le biais d'êtres humains. Aujourd'hui, elles transigent sur des systèmes informatiques automatisés. Il est question de milliers de transactions chaque microseconde. La compétition est féroce et le risque est élevé. Donc, transiger à la bourse implique que les commerçants doivent être préparés. Afin de diminuer le risque de pertes, ces derniers ont recours à des outils de simulations complexes. De plus, il existe une autre forme d'instrument financier moins risquée que les actions: les options. Afin d'augmenter le rendement et de réduire le risque encore plus, il existe plusieurs stratégies d'investissement exploitant les options. L'investisseur doit être en mesure de simuler l'exécution de ses stratégies avant son entrée sur le marché boursier. On dit souvent que le passé est garant du futur. C'est pourquoi l'utilisation de données passées lors des simulations est nécessaire. Ce type de simulation dans le domaine financier est nommé « test rétroactif de validité » ou « backtesting ». Malheureusement, l'accès à une plateforme gratuite de backtesting n'est pas possible.

En réponse à ce problème, l'étudiant Thomas Maketa a mandaté notre équipe pour concevoir une plateforme de backtesting d'options. Celle-ci implique le développement d'une interface web permettant d'écrire des stratégies, effectuer des simulations et visualiser les résultats. Un système de départ fourni doit être adapté aux options et une interface doit être ajoutée.

Ce document a pour but d'expliciter les objectifs de ce projet, la méthodologie employée, les risques technologiques, l'architecture et les caractéristiques de la plateforme web développée, le réusinage effectué sur le code source déjà en place, notre stratégie de tests, ainsi que nos recommandations pour le futur de ce projet.

CHAPITRE 1 - DÉTERMINATION DE L'OBJECTIF DU PROJET

1.1 Contexte et problématique

L'analyse automatique des sentiments est devenue un des domaines importants de l'analyse des langues. En finances, il permet de traiter de grandes quantités d'articles et de dégager un sentiment par rapport à un sujet donné. Les sentiments sont un outil supplémentaire disponible aux stratèges financiers. Ils nous permettent de développer des algorithmes plus évolués pour interagir avec le marché boursier. Interférer avec la bourse n'est pas chose facile. À l'heure actuelle, la majorité des acteurs de ce milieu sont des systèmes informatiques exécutant des algorithmes de trading.

Dans le cadre de ce projet, bien que le marché des actions soit mieux connu, nous nous pencherons sur la gestion des options. À l'heure actuelle, peu de plateformes de trading open source nous permettent de traiter le backtesting des options.

1.2 Objectifs

Dans le cadre de notre projet final, nous souhaitons développer un système informatique de backtesting qui nous permet de traiter des options boursières. L'approche utilisée est de permettre à un utilisateur de développer des stratégies de finance, à l'aide d'investissements et d'outils de décisions, pour générer des profits.

Le projet est essentiellement composé d'un module de backtesting. En effet, ce dernier doit exécuter la stratégie de gestion des options sur un ensemble de données. De plus, il doit offrir à l'utilisateur de consulter et manipuler les données sur les options. Enfin, il y a l'hébergement du précédent projet. Nous optons pour les technologies populaires et éprouvées de la multinationale Amazon.

Jusqu'à présent, nous croyons que le précédent objectif de concevoir un module d'extraction et de stockage d'indicateurs provenant de réseaux sociaux est hors d'atteinte. En effet, l'objectif implique une charge de travail qui dépasse le spectre de notre projet. Le modèle Sentic est un nouveau paradigme d'analyse et la détection de sentiments semble intéressante. Il implique cependant de consacrer beaucoup de temps à peupler un dictionnaire ontologique spécifique au domaine financier. D'ailleurs, cette portion du travail représente davantage un travail d'analyse syntaxique et de grammaire anglaise.

CHAPITRE 2 - APPROCHE

2.1 Méthodologie

Ce projet implique 2 phases de développements qui sont séparés en 10 objectifs. Chaque objectif sera complété par chaque membre de l'équipe dans une approche itérative.

Notre projet implique l'implémentation et la stabilisation de la plateforme de backtesting boursier. Soit les objectifs suivants :

1. Lire et comprendre l'architecture du module de trading financier et ses limitations.
2. Proposer des améliorations et réusiner la plateforme existante pour une meilleure stabilité.
3. Offrir une interface visuelle web pour interagir avec la plateforme.
4. Proposer un plan de test pour s'assurer que les nouvelles fonctionnalités ne causent pas de dommages au système actuel.
5. Ajouter des fonctionnalités à la plateforme PyAlgotrade pour permettre l'utilisation des options.
6. Déployer le projet sur les technologies d'Amazon, plus spécifiquement Amazon Elastic Beanstalk (EBS).

2.2 Techniques et outils

Plusieurs techniques et outils différents ont été utilisés tout au long du projet. En voici une liste exhaustive:

2.2.1 Gestion de code source

- GitHub sera utilisé comme dépôt de code source.
- Un *fork* du dernier projet d'étude est utilisée.

2.2.2 Gestion de projet

- Google Drive pour les documents administratifs.
- GitHub pour la gestion et l'attribution des tâches.
- Rencontres hebdomadaires SCRUM

2.2.3 Outils de développement

- Suite PyCharm de JetBrains pour le développement en Python.
- Utilitaire Meld pour effectuer des comparaisons de code source.

2.2.4 Outils de déploiement

- AWS (Amazon Web Services)
- Amazon EBS (Elastic Beanstalk)

2.3 Livrables et planifications

Pour réussir un projet dans une contrainte de délai, il faut planifier. Notre méthodologie agile SCRUM nous a permis d'ajuster nos jalons à étape du projet. Le tableau ci-dessous présente les différents livrables remis au client.

Nom du livrable	Description	Statut
Diagramme d'architecture sommaire du projet	Architecture du système d'analyse de sentiments en finances et du "trading" en continu.	Terminé
Code réusiné	Réusinage du code déjà en place pour qu'il y ait une extension distincte entre les nouveaux modules et la plateforme PyAlgoTrade.	Terminé
Interface web	Développement de l'interface web qui accède aux fonctions de la plateforme.	Terminé
Plan de test	Tests à effectuer pour valider les fonctions de la plateforme.	Terminé.
Diagramme de déploiement	Représente l'infrastructure physique utilisée par le système ainsi que la répartition des composantes et les relations entre elles.	Terminé
Prototype du système Sentic	Algorithme d'analyse des sentiments du marché de la finance.	Hors d'atteinte
Proposition théorique du modèle d'utilisation de Sentic.	Tactique d'achats et de ventes d'options pour rendre l'algorithme d'analyse des sentiments d'être rentable.	Hors d'atteinte
Développement et test du prototype	Développement et tests de la tactique d'achats et de ventes d'options.	Hors d'atteinte
Intégration plateforme d'échange	Stratégie d'intégration pour PyAlgoTrade.	Hors d'atteinte
Repère indiciel du marché boursier (ex.: S&P500)	Offrir un repère indiciel du marché pour comparer les performances de notre algorithme.	Hors d'atteinte

Voici un Work Breakdown Structure mis à jour par rapport à la proposition de projet pour prendre en compte les modifications de la portée du projet.

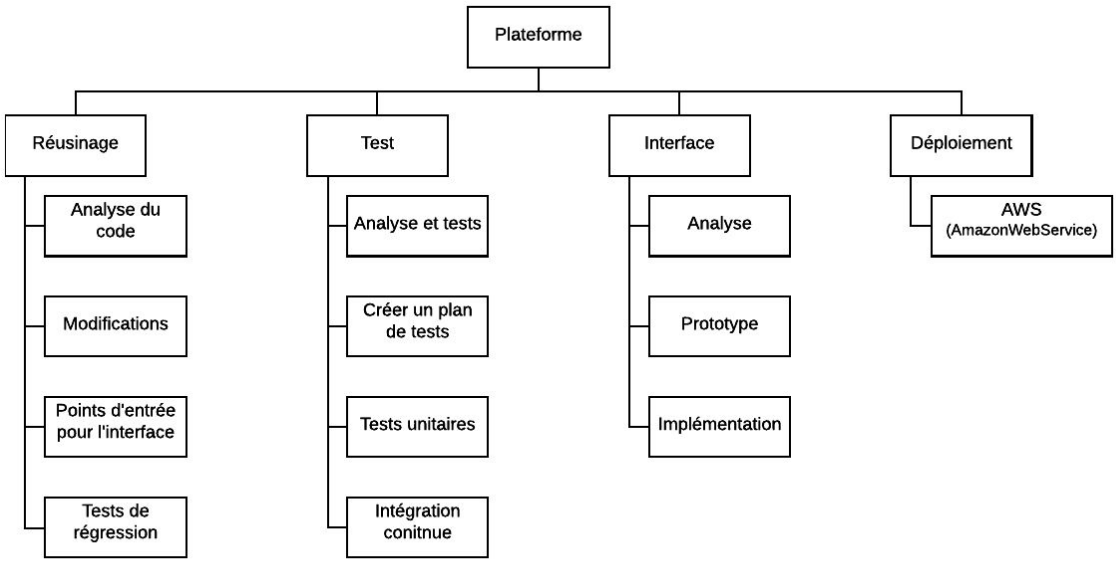


Figure 1 - WBS

Dû à la nature itérative de notre projet l'effort estimé est identifié chaque semaine pour les tâches à venir et l'effort réel peut être vu dans le SCRUM de la semaine suivante. Nous n'avons pas eu de problèmes avec l'approche itérative.

2.4 Risques

La gestion des risques est essentielle. Même si on ne peut pas tout prévoir, on peut les identifier et les prévenir. Le tableau ci-dessous présente les différents risques auxquels nous avons fait face au cours de ce projet et leur impact sur notre développement.

Risque	Impact	Probabilité	Mitigation / atténuation
Ne pas comprendre les concepts des stock-options.	Blocage hâtif.	Faible	Effectuer des lectures sur le domaine et relire nos notes de cours en comptabilité.
Ne pas implanter un protocole de test pour le contrôle de qualité.	Impossibilité de fournir un processus qui garantit que la plateforme fonctionne sans erreur(s) logique(s).	Faible	Revoir les bases de tests et de contrôle qualité des logiciels.

Ne pas comprendre les concepts d'analyse sémantique.	Impossibilité de créer un module d'analyse de sentiments en finance.	Faible	Revoir les bases du NLTK et de l'analyse sémantique pour la grammaire anglophone.
Intégration d'un éditeur de stratégie PyAlgoTrade.	Ralentissement du développement de nouvelles stratégies.	Faible	Étudier des exemples comme Quantopian et effectuer des tests sur un environnement fermé.
Intégration des graphiques amCharts.	Difficulté d'intégration d'un outil d'affichage graphique pour la visualisation des stock-options.	Moyen	Étudier et analyser l'exemple de graphique d'actions sur le site de TickSmith.
Récupération automatisée de données backtesting pour les options.	Impossibilité d'effectuer des opérations boursières les options qui n'ont pas été prévues par les développeurs.	Moyen	Récupération des données à la main sur le site de Yahoo Finances et formatage selon le bon format.
Transfert du data de PyAlgoTrade vers le site web pour les résultats.	Impossibilité d'accéder aux résultats d'une stratégie.	Faible	Analyser et concevoir une méthode de transfert de données du backend vers des graphiques et tableaux.

CHAPITRE 3 - PYALGOTRADE

3.1 Introduction au système PyAlgoTrade

PyAlgoTrade est une librairie, libre d'utilisation, pour le backtesting, le paper-trading et le live-trading d'actions. Elle a été développée par Gabriel Becedillas entre 2011 et 2015. Sa documentation est complète et son utilisation plutôt complexe. Il s'agit d'un système tiers qui facilite la communication avec des courtiers, contient plusieurs modules de comptabilisation de statistiques et facilite l'implantation de stratégies d'investissements avec les actions uniquement. Le logiciel est disponible sur le site suivant : http://gbeced.github.io/pyalgotrade [3]

PyAlgoTrade est un système qui exécute des stratégies utilisateurs de façon cyclique. Ce dernier reçoit des données dans un intervalle de temps fixe. L'utilisateur peut prendre des décisions en fonction des données qu'il reçoit à chaque intervalle, soit vendre et acheter des instruments financiers.

3.1.1 Pourquoi PyAlgoTrade?

PyAlgoTrade a fait l'objet d'une analyse poussée par Nicolas Hubert, un étudiant d'une session antérieure. Le système a été décortiqué pour en faire sortir ses avantages et inconvénients. Parmi les avantages, on retrouve notamment la facilité d'ajout de nouveaux modules pour les stratégies d'options sans modifier le code existant. Les principes d'héritage et d'encapsulation sont utilisés dans l'ensemble du projet, ce qui facilite les modifications. L'adaptation au format de la source de données est simple à modifier. Davantage d'explications concernant le choix de PyAlgoTrade sont fournies dans le rapport technique de Nicolas.

CHAPITRE 4 - RÉUSINAGE

4.1 Motivations

Une des préoccupations avec le code récupéré du dernier projet est que les nouvelles classes d'options étaient dans les mêmes modules que les actions. C'est une bonne pratique en développement logiciel de ne pas modifier des composantes inutilement, mais plutôt encapsuler le comportement de nouveaux modules. Ceci permet d'éviter des impacts collatéraux sur le code existant qui est fonctionnel et testé selon son état initial.

4.2 Stratégie

Le code a d'abord été analysé afin de comprendre le comportement général du système. Les efforts ont été concentrés sur les intrants et les extrants. Ensuite, dans le but d'identifier les classes affectées par les modifications du projet précédent, un outil de comparaison de code source est utilisé. En effet, en comparant le code source de PyAlgoTrade et la version modifiée du dernier projet, les fichiers concernés par l'ajout des classes d'options ont été isolés.

4.3 Modifications

Dans le but de supporter les options, PyAlgoTrade a subi plusieurs modifications pour atteindre les objectifs du mandat. De nombreux modules ont été déplacés afin d'augmenter la cohésion de ce projet. Une problématique a été détectée au niveau des sources de données. Celles-ci ne sont pas conformes aux données d'options boursières standards.

Suite au réarrangement des classes d'options dans le code de PyAlgoTrade, la solution finale présente davantage de cohésion. En effet, les classes utilisées pour les options sont placées dans des dossiers distincts à l'intérieur d'un *package*. Les diagrammes de classes pour les modules *Order, Broker, Position* et *Strategy* sont disponibles sur la prochaine page.

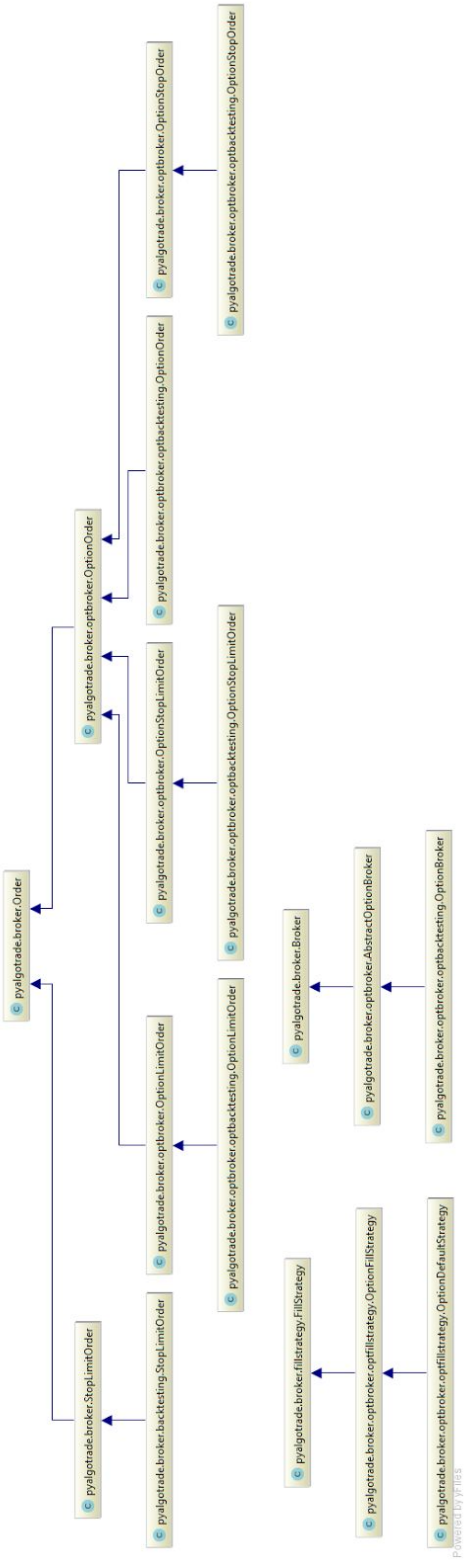


Figure 2 - Diagramme de classes de Broker

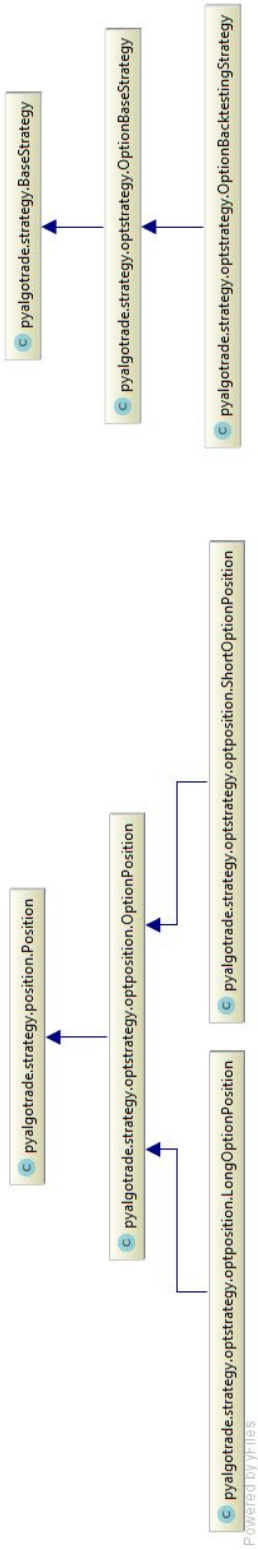


Figure 3 - Diagramme de classes de Strategy

CHAPITRE 5 - NOTRE PLATEFORME

5.1 Besoins

Exécuter des stratégies financières nécessite beaucoup de traitement. Celles-ci doivent être exécutables le plus rapidement possible. La rapidité est un facteur déterminant sur le marché financier. Effectuer plusieurs opérations dans un court laps de temps offre un avantage par rapport aux compétiteurs. Par exemple, un trader pourrait acheter et vendre 100 actions, alors que son compétiteur n'a eu le temps que d'en échanger 5. La simplicité fut le second facteur déterminant dans l'élaboration de notre plateforme. Seuls les éléments essentiels à la rédaction de stratégie financière sont présents. Ils sont placés de façon à optimiser l'espace d'affichage. Il s'agit d'une réplique simplifiée d'une interface de développement pour programmeur. Des graphiques ont été placés afin de visualiser les informations contenues dans les sources de données.

5.2 Inspirations

Une de nos principales inspirations est le site web Quantopian. Ce site web est utilisé pour effectuer des stratégies financières uniquement liées aux actions. Les options ne sont pas supportées. La disposition des éléments de ce site web nous a fortement inspirés, ainsi que la façon de l'utiliser. En effet, les diverses fonctionnalités offertes à l'utilisateur lui permettent de récupérer des données historiques, exécuter une stratégie de backtesting et consulter le rendement monétaire. L'information est présentée de manière conviviale. En effet, les données historiques, ainsi que les évènements de vente ou achat d'actifs sont présentés dans un graphique interactif AmCharts. De plus, l'utilisateur peut consulter le rendement de sa stratégie, une fois que son algorithme est exécuté, dans un sommaire des résultats.

5.3 Choix technologiques

Plusieurs technologies sont disponibles pour combler les besoins de notre plateforme. Tout d'abord, le cadriciel Django. Celui-ci est basé sur le patron architectural MVC et est basé sur le langage de programmation Python. En plus d'être compatible avec le même langage de programmation, celui-ci offre l'implémentation rapide d'interfaces utilisateurs avec Bootstrap. Il est important de savoir qu'une version de Bootstrap est préparée spécialement pour Django.

Les technologies AmCharts nous ont permis d'afficher les données d'options sur des graphiques interactifs. Elles sont basées sur les langages JavaScript et Html5 en plus d'être interactives, personnalisables et très bien documentées. Ils automatisent beaucoup

d'opérations et gèrent de nombreuses caractéristiques financières. En effet, lorsque les données sont fournies, il est possible d'ajuster le zoom, sélectionner un élément pour obtenir plus d'informations et d'afficher ou non les événements de ventes et achats suite à l'exécution d'une stratégie.

L'éditeur de texte Ace nous est apparu comme la solution la plus simple, rapide et efficace à intégrer pour notre plateforme. En effet, ce dernier est personnalisable, rapide à charger et offre des fonctionnalités comme l'intellisense. Cependant, cette fonctionnalité n'est pas exploitée pour le moment. Son utilisation le rend encore plus attrayant. Une seule ligne de code est nécessaire pour récupérer son contenu. Ce contenu est ensuite envoyé vers le serveur pour subir une analyse et le traitement adéquat.

5.4 Exécution de code source

La fonction principale de notre application web est de pouvoir écrire le code d'une stratégie financière, l'exécuter dans PyAlgoTrade et de récupérer le résultat. Ceci représente un obstacle puisque le code source doit être compilé et exécuté pendant l'exécution même de l'application web. Heureusement, la librairie de Python offre plusieurs fonctions pour atteindre ce but. En effet, les fonctions *compile(...)* et *exec(...)* permettent respectivement de compiler et d'exécuter un script Python comme si ce code faisait partie de l'application lors de l'appel. Nécessairement, les modules de PyAlgoTrade doivent être visibles dans le *namespace* de l'application courante pour que tout fonctionne.

Cependant, la récupération du résultat ne peut pas se faire à travers le mécanisme de retour traditionnel de Python puisque *exec(...)* retourne *None*. Donc, une stratégie de transfert d'informations d'exécution a été conçue. Essentiellement, pendant l'exécution du code de PyAlgoTrade, le résultat de backtesting d'une stratégie est sérialisé vers un fichier. Lorsque le contexte revient vers l'application web, ce même fichier est lu et le contenu est envoyé à la page web pour être affiché.

Enfin, puisque le code que nous exécutons n'est pas nécessairement fiable, nous avons adopté une stratégie plutôt simple pour empêcher des opérations non désirables. Avant que le code soit compilé, il est balayé afin de détecter toute opération qui pourrait être d'intention malicieuse. On parle ici d'opérations telles que: *import* pour l'inclusion de module quelconque et *open* pour ouvrir, par exemple, un fichier critique du système d'exploitation. Puisque l'utilisateur ne peut pas importer de modules, tous les modules dont il a besoin pour

écrire une stratégie sont visibles lors de la compilation. S'il désire ajouter un module, il doit faire la demande aux développeurs de l'application.

5.5 Source de données

Les sources de données de notre application sont les informations d'options boursières. Présentement, la version de notre système ne supporte que les données liées aux actions. Le système actuel présente des informations qui se traduisent par: *open*, *close*, *high*, *low*, *number of ticks*, *volume* et *value*. Une modification est requise pour permettre d'afficher les informations d'options sur le graphique AmCharts et les rendre accessibles à la stratégie de l'utilisateur. Chaque entrée de donnée pour une option contient les informations suivantes: *last price*, *bid*, *ask*, *change*, *volume*, *open interest* et *implied volatility*. Ces informations doivent être ajoutées à des fichiers de données respectant une nomenclature précise dans le dossier "Data" situé à la racine de l'arborescence du projet. La figure ci-dessous illustre la nomenclature à respecter:

Figure 4 - Dossier de données

bac: instrument (Bank of America Corporation)
2000: *Strike price* (2000 pour 20.00$, 017 pour 0,17$ ou 14350 pour 143.50$)
p: Type d'option ("p" pour un *PUT* et "c" pour un *CALL*)
20160819: *Expiry date*

Les sources de données actuelles sont présentées à l'utilisateur dans une liste déroulante. Cette méthode a été choisie en raison du nombre limité de fichiers de données. Ultérieurement, il sera impératif d'avoir des champs de sélection séparés pour chaque information d'option: *strike price*, *put/call*, *expiry date*. Cela permettra d'afficher seulement les jeux de données pertinents.

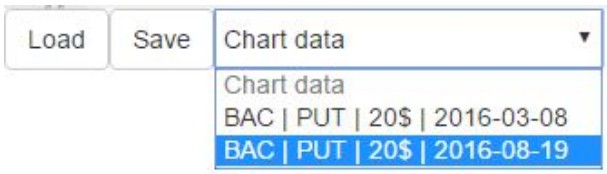


Figure 5 - Sélection de la source de données

5.6 Déploiement

Puisque nous voulions un processus de déploiement simple, rapide et efficace, nous avons opté pour la technologie Amazon Web Services (AWS); plus spécifiquement, Amazon Elastic Beanstalk. Un guide de déploiement est inclus à l'Annexe I. Voici un aperçu de la structure lorsque la plateforme complète est déployée:

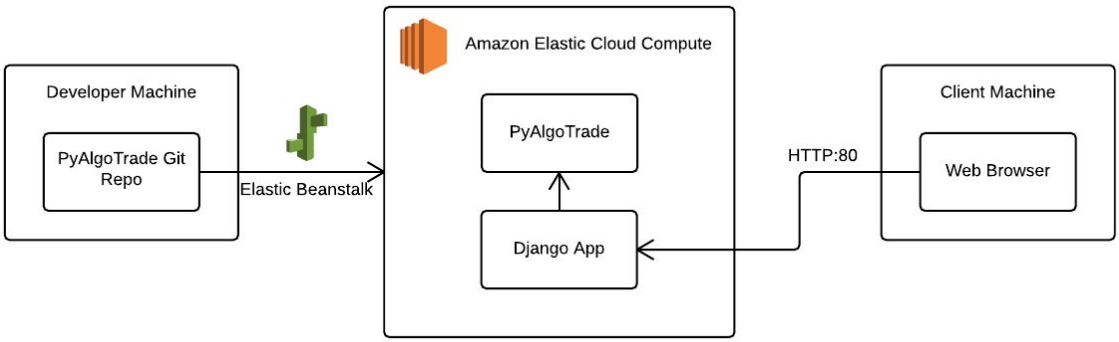


Figure 6 - Diagramme de déploiement

5.7 Rédaction de stratégies boursières

Suite au déploiement de l'application web sur un serveur Amazon Web Services Beanstalk, il est désormais possible d'écrire des algorithmes de stratégies financières avec le langage Python. Celles-ci ne sont pas enregistrées sur le serveur puisqu'elles doivent demeurer confidentielles. Il est actuellement possible de simuler la vente et l'achat d'actions et d'options pour le *backtesting*. De plus, il est important de mentionner que les données d'options doivent être ajoutées à la main dans le dossier "Data" prévu à cet effet. Ce dernier est situé à la racine du projet. Le format de fichier CSV doit être en règle: un nombre de colonnes fixes à chaque entrée de données (ligne).

Plusieurs algorithmes de base existent déjà dans le domaine de la finance. Des indicateurs permettent de faciliter la gestion d'un portefeuille. Parmi ces indicateurs, on retrouve le "Single Moving Average". C'est d'ailleurs cette mesure qui est utilisée dans la stratégie présente par défaut sur le site web. Les stratégies payantes ne se trouvent pas librement sur internet. Afin de générer des sommes d'argent, l'utilisateur doit utiliser de sa créativité et innover. Toutefois, certaines techniques de base permettent à l'utilisateur de s'exercer: *Covered Call*, *Butterfly*, *Bear put spread*, *Strangle*. Ces 4 stratégies nous ont été présentées par Thomas Maketa dans le cadre de ce projet.

CHAPITRE 6 - TESTS

6.1 Objectif des tests

L'application PyAlgoTrade ayant été modifiée, il fallait maintenant s'assurer que le tout était encore fonctionnel. Tester une application qui traite l'argent des utilisateurs est très important. Présentement, PyAlgoTrade permet de faire des transactions sur les actions en temps réel. L'application se doit d'avoir été testée en profondeur pour ne pas avoir de bogues qui pourraient coûter très cher en argent. Une erreur pourrait vider totalement votre portefeuille d'argent et d'actions ce qui serait catastrophique. La version modifiée pour les options de PyAlgoTrade ne permet pas encore de faire des transactions en temps réel avec de la vraie monnaie, mais ceci est définitivement une fonctionnalité intéressante à implémenter dans le futur de l'application. C'est pourquoi des tests ont été implémentés pour avoir une bonne qualité de code et permettre une meilleure transition dans le futur.

6.2 Stratégie de test

Comme cela a déjà été mentionné, faire des transactions non annulables en temps réel avec de la vraie monnaie implique que PyAlgoTrade se devait d'être de très bonne qualité. Les créateurs ont effectivement implémenté une stratégie de test qui consiste à tester chaque classe et chaque méthode de manière exhaustive à l'aide de tests unitaires. Les tests sont très complets et vont même jusqu'à s'assurer que les exemples du tutoriel sont fonctionnels. La stratégie utilisée pour tester nos modifications est de prendre les tests déjà existants sur les classes que nous avons modifiés et les copier dans un dossier de test pour options et les adapter aux changements. Pour chaque nouvelle fonctionnalité ajoutée, un nouveau fichier de test est créé et celui-ci doit tester en profondeur la classe et les méthodes.

6.3 Implémentation des tests et bogues trouvés

Chaque classe créée devait être testée même si aucune fonctionnalité n'avait été ajoutée et la totalité des changements n'était que des changements de noms. Plusieurs classes étaient dans un script Python. Comme PyAlgoTrade, un seul script de test par script Python était créé. Les fichiers à tester en adaptant les tests déjà existants sont : *ibfeed*, *optbroker*, *optbacktesting*, *optfillstrategy*, *optstrategy* et *optposition*. La classe ajoutée qui contient une nouvelle fonctionnalité à tester est filename. Cette série de tests permet de tester le bon

fonctionnement de base des nouvelles classes avec leur nouveau nom et leurs nouveaux paramètres qui ne sont pas encore utilisés.

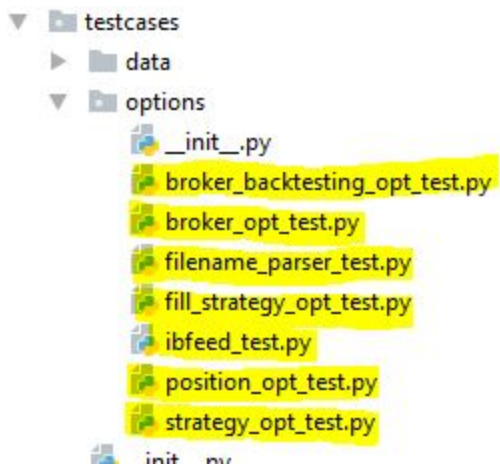


Figure 7 - Arborescence de testcases

6.4 Résultat des tests

L'implémentation des tests nous a permis de trouver plusieurs problèmes. Un des plus importants était l'utilisation de la nomenclature "optionOrder" au lieu de "optionMarketOrder". Normalement, la classe "Order" était héritée par "marketOrder", "limitOrder", "stopOrder" et "stopLimitOrder". Dans nos modifications, la classe "optionOrder" agissait comme la classe "marketOrder" et était héritée par les trois autres. La classe parente "Order" était manquante. Le même problème de confusion entre "optionOrder" et "optionMarketOrder" a eu des répercussions sur plusieurs autres classes et chaque problème a été détecté par les tests.

Un autre bon résultat des tests est la découverte de plusieurs problèmes avec notre script filename qui n'était pas utilisé, mais qui devait, en théorie, gérer la nomenclature des fichiers Excel et en extraire les informations. Le script pouvait gérer seulement les symboles financiers de trois lettres. Par exemple le symbole très populaire "APPL" ne fonctionnait pas. Le script acceptait aussi que deux chiffres pour le *strike price* et ne gérait pas les décimales. Ces bogues ont été trouvés avec les tests écrits pour le script *filename*.

6.5 Changements dans le futur

Les tests créés dans pour la partie options de l'application sont en majorité des copies des tests déjà présents dans PyAlgoTrade avec les nouvelles classes utilisées et les nouveaux noms de méthode aux classes changées. Le seul autre changement est l'ajout des paramètres *right*, *strike* et *expiry* aux endroits nécessaires. Présentement, ces nouveaux paramètres ne sont pas utilisés dans les classes. Dans le futur, après l'ajout de fonctionnalités pour les options plus détaillées, ces classes devront être revisitées pour ajouter des scénarios utilisant les nouvelles fonctionnalités. Par exemple, une de ces fonctionnalités est la gestion de l'expiration qui devrait être testée très exhaustivement une fois ajoutée. Aussi, tous les tests utilisent les mêmes trois paramètres. Ceci pourrait créer des problèmes quand ceux-ci seront utilisés dans le futur. Il faudra repasser dans les tests et s'assurer que tous les cas fonctionnent.

CHAPITRE 7 - EFFORTS INDIVIDUELS

7.1 Répartition de la charge de travail

Le tableau ci-dessous présente la répartition de la charge de travail sur les différents modules implémentés dans le cadre de ce projet final pour chaque étudiant qui a participé.

Module / Apport	**Benjamin**	**David**	**Sébastien**
Réusinage	33%	33%	33%
Intégration de tests	100%	0%	0%
Interface utilisateur	0%	60%	40%
Logique d'affaires	33%	33%	33%
Personnalisation graphique	0%	40%	60%
Gestion de projet	33%	33%	33%
Documentation	33%	33%	33%
Moyenne	**~33%**	**~33%**	**~33%**

Somme toute, l'apport moyen pour chaque étudiant dans ce projet a été réparti uniformément. Bien que ces chiffres illustrent l'ensemble de l'effort individuel, il reste tout de même des heures supplémentaires à ajouter à chaque module. En effet, l'interface utilisateur aura besoin de peaufinage, les graphiques devront être mieux adaptés aux données d'entrées et l'exécution de stratégie devra se faire plus rapidement. D'ailleurs, les recommandations sur ce travail sont présentées dans le prochain chapitre.

CHAPITRE 8 - FUTUR DU PROJET

8.1 Limitations

8.1.1 Opérations sur les options

Une des principales limitations de projet est l'implémentation complète de la gestion des options. Malheureusement, nous n'avons pas pu intégrer les fonctionnalités de papier et de live trading. Bien que la plateforme PyAlgotrade supporte les opérations de base pour les actions, un niveau d'abstraction a été ajouté pour différencier les opérations sur les options et les actions. Cette méthode permet une distinction des instruments dans notre portfolio. Cela ouvre la porte aux modifications futures. Bien sûr, nous voulons éviter de modifier les classes originales du projet. De plus, pour exécuter un *backtesting* de stratégies sur des options, il manque l'implémentation de gestion d'options plus complète. L'utilisateur doit être capable d'écrire des options, c'est-à-dire créer des contrats d'options et collecter la *prime* de ventes de ceux-ci. Enfin, il manque la gestion de l'expiration des contrats d'options. Celle-ci est actuellement gérée au niveau de la stratégie utilisateur.

8.1.2 Données avec plusieurs *strike price* et *expiry date*

Présentement, l'application n'a pas d'architecture permettant une gestion simple et facile de données d'options pour l'utilisateur. Les données d'un contrat d'options sont très similaires aux données d'une action, elles contiennent un prix, un *bid,* un *ask*, etc. La différence est dans la quantité de contrats possible. PyAlgoTrade a été conçu pour faire du backtesting sur un petit nombre de symboles. Par exemple, une stratégie prenait majoritairement un seul symbole, comme "APPL", et exécutait le *backtesting* sur ce symbole seulement. L'architecture permet plusieurs symboles ou plusieurs *feed* mais seulement un petit nombre. Le *feed* passé en paramètre à la stratégie est un tableau qui peut contenir plusieurs symboles, mais par la suite si l'utilisateur veut faire un calcul sur chaque symbole, il n'y a pas d'architecture intégrée. Il faut faire une ligne par symbole dans le *feed*. L'architecture avait été pensée pour un petit nombre de symboles et ceci ne fonctionne pas avec les options qui pourraient avoir une quantité énorme de contrats dans le *feed*. Dans le monde des options, à chaque *bars* (minute, seconde, etc) le seul symbole "APPL" peut avoir dans les environs de 16 *expiry date* avec chacun 60 *strike price* différents ce qui donne 960 contrats potentiels à acheter, vendre ou écrire. L'application actuelle oblige l'utilisateur à implémenter lui-même cette gestion dans sa stratégie.

8.1.3 Source de données

La technique actuelle de gestion des données n'est pas très flexible. En effet, l'utilisation des fichiers CSV doit être une technique temporaire si l'objectif est d'effectuer du *backtesting* de qualité. Bien que l'on puisse tester sur des données historiques, il est impératif que les stratégies développées soient évaluées sur des données actuelles, ce qui est actuellement impossible.

8.1.4 Interface

On retrouve une limitation fonctionnelle de l'interface utilisateur. Lors de l'exécution de *backtesting* d'une stratégie, l'interface est gelée en attente du résultat. Cette approche n'est pas conviviale puisque l'interactivité avec l'utilisateur est coupée. Ce serait préférable s'il y avait un retour visuel périodique pour ne pas perdre l'attention de l'utilisateur.

8.2 Recommandations

8.2.1 Opérations sur les options

Il serait primordial d'intégrer les opérations sur les options dans la solution actuelle. Par exemple, celles-ci devraient être achetées ou vendues en fonction des paramètres comme le volume d'une option. Il devrait aussi être possible d'écrire des options. La gestion d'expiration devrait être automatisée. Nous pensons que le module *optBroker* serait le plus susceptible d'accueillir cette logique. À savoir que la logique peut être intégrée à multiples endroits dans le code. Des exigences peuvent être trouvées dans la section 8.3.

8.2.2 Données avec plusieurs *strike price* et *expiry date*

Après avoir intégré les opérations sur les options et la gestion d'expiration, le plus important serait d'implémenter une gestion des données d'options. Ceci serait fait selon la source de données utilisée ou les sources de données utilisées. Nous suggérons l'utilisation d'une seule source pour débuter. Si la source de données actuelle de fichiers Excel est conservée, il faudrait avoir un fichier Excel pour chaque contrat et permettre aux stratégies de pouvoir comparer des contrats. Par exemple, pour savoir lequel serait le meilleur à acheter pour avoir un profit maximal en suivant une stratégie quelconque. Une implémentation flexible selon la source de données serait l'idéal, mais pourrait être beaucoup plus compliquée à concevoir.

8.2.3 Source de données

Puisque le fait de supporter les options engendre beaucoup de données, il serait intéressant de pouvoir permettre à l'utilisateur de limiter le nombre de fichiers offerts à lui en ajoutant des champs de recherche tels que: instrument, type, strike price et expiry date. De cette manière, l'utilisateur n'aurait pas à perdre son temps à chercher dans la liste déroulante de lots de données.

En parallèle, l'utilisation de fichiers csv comme source de données n'est pas une approche à long terme. En effet, chaque ajout de fichier engendre des opérations supplémentaires de disque coûteuses. Voici deux solutions alternatives qui seraient mieux adaptées à ce type de problème.

Une première approche pour solutionner ce problème serait d'utiliser une base de données afin de stocker les données d'options. En termes de technologies, soit une base de données relationnelle ou *NoSQL* serait convenable. Il n'y a pas de besoins pour l'analytique donc le *big data* n'est pas tant pertinent dans cette situation. Pour ce faire, il suffirait d'écrire un script d'importation de csv vers l'engin de base de données sélectionné. Bien que l'accès aux données serait généralisé, cela n'enlève pas le problème de l'ajout de nouvelles données.

L'approche recommandée serait d'utiliser un API externe afin de récupérer les données d'options. En utilisant les champs de recherche décrits dans la section précédente, un appel vers un service web serait effectué facilement. De cette manière, on se sépare de la responsabilité d'avoir à gérer les données et on sait que les données sont toujours à jour. De plus, puisqu'il ne serait plus nécessaire d'importer chaque fichier de données, il serait possible, voire simple, de pouvoir effectuer du backtesting sur des lots de données complètement différents. Ceci ferait en sorte que les algorithmes développés soient le plus robustes possible. Il est impératif que cette approche soit utilisée pour commencer à supporter les opérations sur les données en temps réel.

8.2.4 Interface

Pour faire en sorte que l'application web soit plus interactive, il serait intéressant d'utiliser une technologie basée sur les *websockets* pour afficher les résultats de backtesting. En effet, avec cette approche, les résultats s'afficheront dans l'interface au fur et à mesure que le *backtesting* progresse. Dans le cas où cette approche est adoptée, des changements sont

nécessaires au niveau du code de PyAlgoTrade afin d'établir une connexion via *websocket* avec l'application web.

8.3 Exigences pour supporter les options

La première étape serait de permettre la création de CALL et de PUT. Pour ce faire, une position négative sur l'option est créée. Le *premium* de l'option vendue est collecté à l'expiration ou à l'annulation de la position. Un paramètre qui bloque la création de *naked call* ou de *naked put* pourrait être implémenté pour s'assurer que quand des options sont créées *naked* l'utilisateur en est conscient.

La gestion de l'expiration se fait pour les positions positives et négatives. À la date d'expiration de n'importe quel type de contrat *out-of-the-money*, la position est fermée et le *premium* est reçu si c'était un contrat créé par l'utilisateur (une position *short* ou négative). Si le contrat est *in-the-money,* il est exercé que ce soit une position négative ou positive. Si le portfolio contient une position de -1 sur un CALL ou de 1 sur un PUT *in-the-money,* celui-ci est exercé et le portfolio doit vendre 100 actions du *stock* au *strike price*. Avec une position de -1 sur un PUT ou une position de 1 sur un CALL *in-the-money*, nous sommes forcés d'acheter 100 actions du *stock* au *strike price*.

Une description détaillée du fonctionnement de l'expiration peut être trouvée dans la documentation de plusieurs *broker* sur Internet comme celui de simulation de la Bourse de Montréal. [2]

CONCLUSION

Somme toute, nous avons conçu une preuve de concept d'une plateforme de backtesting d'options. Les objectifs de ce projet ont été explicités, la méthodologie employée, les risques technologiques auxquels nous avons fait face, l'architecture de notre solution, les caractéristiques de la plateforme web développée, le réusinage sur le code source de la plateforme existante, notre stratégie de tests ainsi que nos recommandations pour le futur de ce projet. Enfin, notre équipe a conçu une interface et y a intégré la plateforme de backtesting existante nommée "PyAlgoTrade". L'utilisateur peut désormais composer avec un graphique de type AmCharts pour visualiser les actions, options, buy et sell, un éditeur de texte pour écrire des algorithmes de stratégies boursières en Python et une console de résultats pour obtenir des informations sur l'exécution de ses requêtes au serveur. Bien que la plateforme possède, désormais, une interface, bien des étapes sont nécessaires à la complétion de ce projet. On parle entre autres de sécurisation du site web à d'autres utilisateurs, la sécurisation du code de la stratégie et une nouvelle façon de récupérer des données d'options. Nous croyons que cette plateforme a beaucoup de potentiel. Dans un avenir proche, nous croyons qu'intégrer nos recommandations à la présente solution ajouterait beaucoup de valeur. En effet, supporter l'écriture d'options, utiliser un API pour obtenir les options et intégrer des websockets pour améliorer l'interactivité de l'interface lorsque la stratégie financière est exécutée.

RÉFÉRENCES

1. Banque de Montréal. 2017. Guide du calculateur
   https://www.m-x.ca/f_publications_fr/calc_guide_fr.pdf
2. Banque de Montréal. 2017. Guide de simulateur
   https://www.m-x.ca/f_publications_en/sim_guide_en.pdf
3. Gabriel Becedillas. 2017. PyAlgoTrade. Version 0.18. Répertoire GitHub.
   http://gbeced.github.io/pyalgotrade/

BIBLIOGRAPHIE

Sale, David. (2014). “Testing Python” [livre électronique]. http://it-ebooks.info/book/3778/

Alain April. 2017. “Projets de recherche en analyse des réseaux sociaux/Wiki's et de logiciel libre”, consulté le 24 Janvier 2017.
https://www.etsmtl.ca/Services/ProjetsRDGelog/Projets/reseaux-sociaux

Nicolas Hubert. 2016. Rapport technique PFE. Version 1.0. Publications Publicationslist.
http://publicationslist.org/data/a.april/ref-567/Hubert_Nicolas_GTI792_A16_v2-3.pdf

Yahoo Finance. 2017. Yahoo Finance Options. https://ca.finance.yahoo.com/

Andreessen Horowitz, Bessemer Venture Partners, Khosla Ventures, and Spark Capital. 2017. Quantopian. Plateforme de développement. https://www.quantopian.com/

Guide de déploiement

pour

PyAlgoTrade

Version 0.1

Rédigé par:
David Méthot
Sébastien Santerre
Benjamin Lebois

École de Technologie Supérieure

11 Mars 2017

Suivi des changements

A* - Ajouté **M** - Modifié **S** - Supprimé

Numéro de version	Date aaaa/mm/jj	Numéro de figure, table ou section	A* M S	Brève description du changement	Numéro de demande de changement
0.1	2017/03/11		A*- Tout	Version initiale du document	

Liste des figures

Nom de la figure **Page**

Figure 1 - Diagramme de déploiement 1

1.0 Déploiement sous AWS

Les étapes suivantes assument que le code source de PyAlgoTrade est déjà présent sur votre poste de travail et que vous avez les droits de création d'environnement Elastic Beanstalk sur votre instance AWS.

1.1 Diagramme de déploiement PyAlgoTrade sous AWS

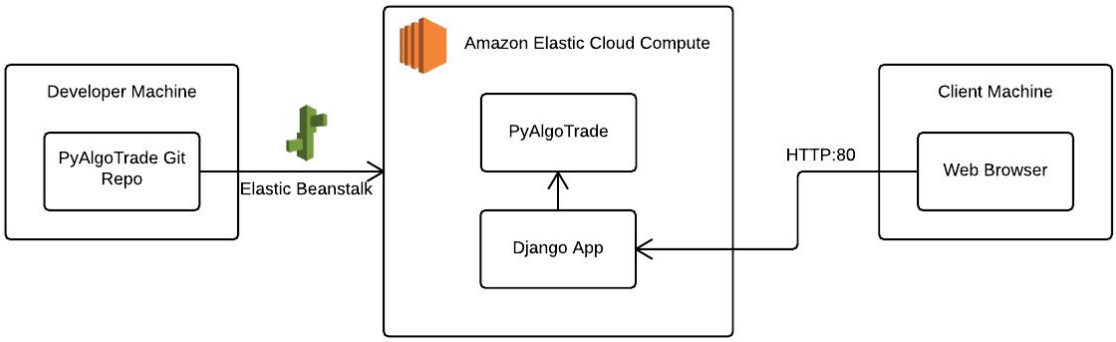


Figure 1 - Diagramme de déploiement

1.2 Installation et configuration du EB CLI

Le Elastic Beanstalk Command Line Interface (EB CLI) est un utilitaire en ligne de commande qui permet de créer, configurer et gérer des environnements Elastic Beanstalk d'Amazon. Avant de poursuivre, assurez-vous d'avoir une installation de Python d'une version minimale de 2.7 avec une installation de pip ainsi qu'une installation de ssh-keygen.

1.2.1 Installation EB CLI

Pour effectuer l'installation du EB CLI, il suffit de suivre les étapes documentées sur le site d'Amazon. Voici l'hyperlien permettant d'accomplir la tâche:

Hyperlien pour installation du EB CLI

1.2.2 Configuration EB CLI

Il s'agit maintenant de configurer votre installation de EB CLI afin de pouvoir déployer des environnements Elastic Beanstalk. Pour les commandes qui suivent, vous devez être à la racine du repository Git de PyAlgoTrade sur votre poste de travail. Suivre les étapes de la section Configuration de l'interface de ligne de commande EB sur la page web suivante:

Hyperlien pour configuration du EB CLI

Si vous éprouvez des difficultés à créer votre *Access Key/Secret Key*, suivez les étapes documentées sur la page web suivante:

Hyperlien pour création *Access Key/Secrey Key*

1.3 Déploiement

Les étapes suivantes assument que le EB CLI a été installé et configuré.

Il est maintenant venu l'étape de déployer PyAlgoTrade sur Elastic Beanstalk. Encore une fois, vous devez être à la racine du repository Git de PyAlgoTrade sur votre poste de travail. Suivre les étapes de la section Déploiement de votre site avec l'interface de ligne de commande EB (normalement vous pouvez sauter au point 3) sur la page web suivante:
Hyperlien pour déploiement d'un projet Django avec EB CLI

Compte-rendus de réunions

Projet

Participant(s)	Sébastien Santerre, étudiant ÉTS Benjamin Lebois, étudiant ÉTS David Méthot, étudiant ÉTS Thomas Maketa, étudiant à la maîtrise ÉTS Alain April, Professeur génie logiciel ÉTS Patrick Cardinal, Professeur génie logiciel ÉTS
Titre	Analyse de sentiments en finances

Table des matières

Procédure d'ajout d'un nouveau compte-rendu 2

Scrum du vendredi, le 20 janvier 2017 3

Scrum du vendredi, le 27 janvier 2017 4

Scrum du vendredi, le 3 février 2017 8

Scrum du vendredi, le 10 février 2017 10

Scrum du vendredi, le 17 Février 2017 14

Scrum du vendredi, le 3 Mars 2017 17

Scrum du vendredi, le 10 Mars 2017 21

Scrum du vendredi, le 17 Mars 2017 24

Scrum du vendredi, le 24 Mars 2017 28

Scrum du vendredi, le 31 Mars 2017 29

Scrum du vendredi, le 7 Avril 2017 30

Scrum du vendredi, le 20 janvier 2017

Rencontre:

- Lieu : ÉTS, 13h00
- Personnes présentes:
    - Alain April, ÉTS
    - Patrick Cardinal, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: David Méthot

Ce qui a été fait:

- Rédaction de la proposition de projet
- Brainstorming du projet, portée, technologies
- Documentation des décisions par rapport au brainstorming
- Signature des documents administratifs
- Démo Ticksmith

Éléments bloquants:

- Portée du système Sentic. Proposition de notre idée préliminaire.
- Qu'est-ce qu'un livrable pour vous? Granularité?

Questions:

- Déploiement AWS nécessite-t-il une modification du code de la plateforme?
- UI supporte paper trading?
- Pertinence de l'Annexe A - Plan de travail

Plan d'actions:

- Création et assignation des tâches dans l'outil de gestion de projet GitHub
- Début des tâches. Planifiées: Analyse du code et début réusinage.

La prochaine rencontre est prévue pour le **<u>27 janvier prochain</u>**.

Scrum du vendredi, le 27 janvier 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Patrick Cardinal, ÉTS
    - Thomas Maketa, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS

Ce qui a été fait:

Sébastien

- Modification proposition de projet. (Problématique, Objectifs, Planification, Techniques et outils)
- Création des tâches dans GitHub pour le suivi de projet.
- Configuration environnement de développement.

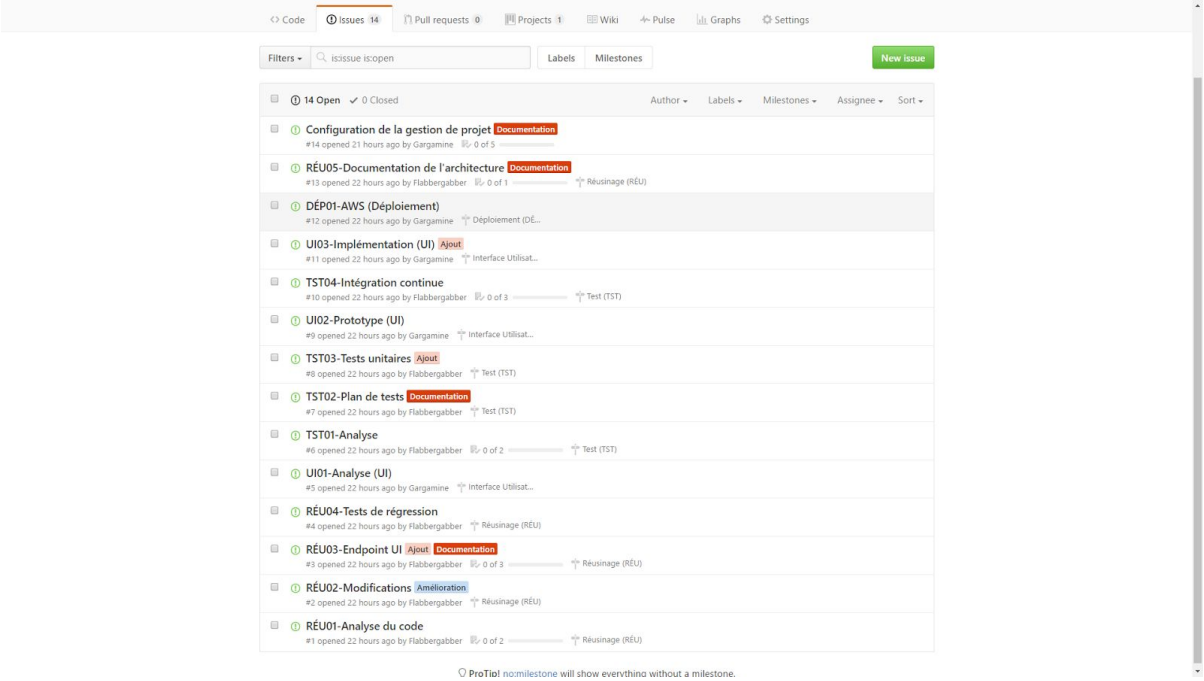
Benjamin

- Analyse du code PyAlgoTrade
- Clean code
- Lire et apprendre plus sur les options

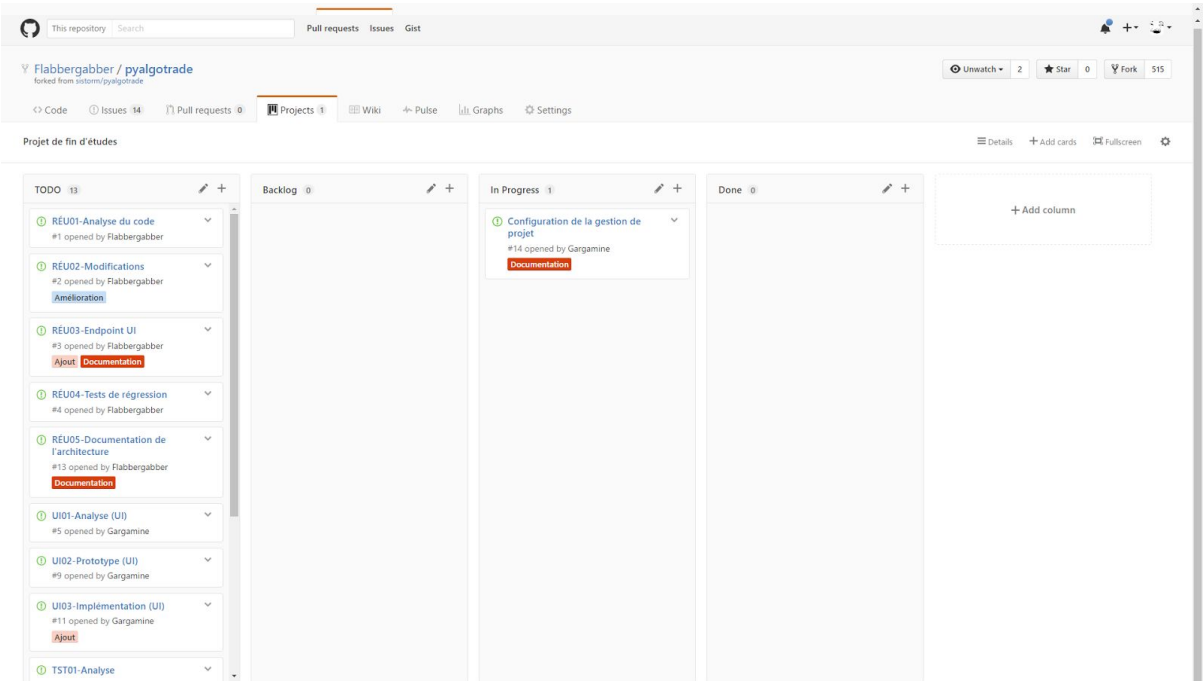
David

- Modification de proposition de projet (Portée, Objectifs)
- Création de la gestion de projet dans GitHub (Issues)
- Recherche et lecture sur Apache Camel
- Mieux comprendre Put/Call options

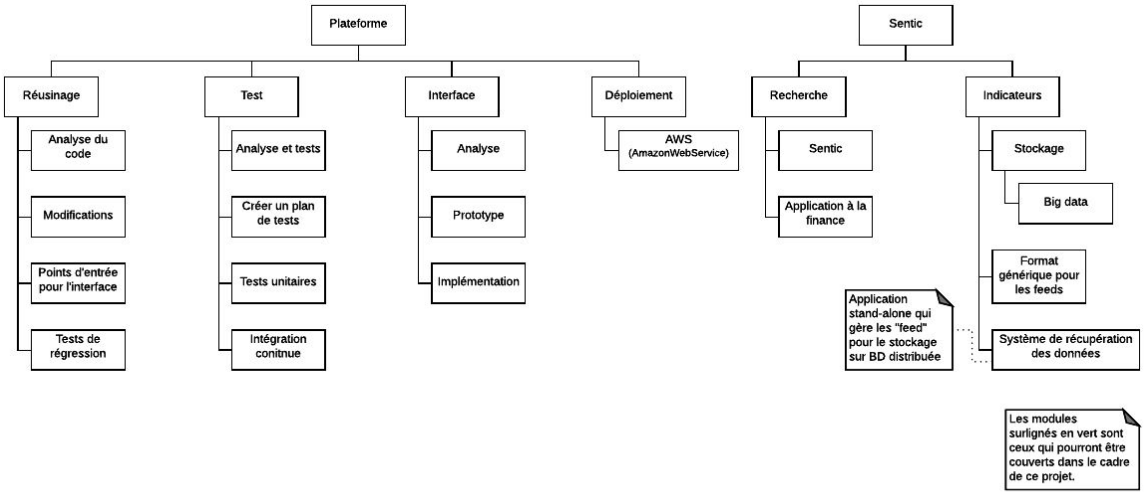
Tâches dans GitHub:



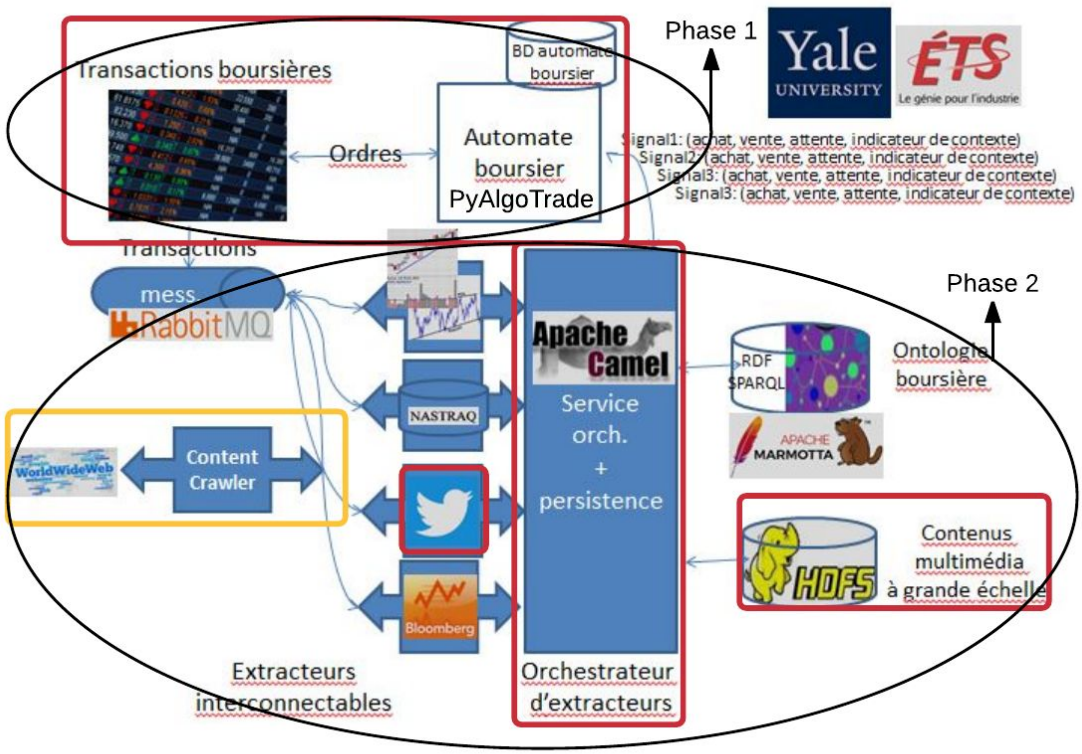
Planning TODO, Backlog, In Progress, Done



WBS


Plateforme
Réusinage
Analyse du code
Modifications
Points d'entrée pour l'interface
Tests de régression
Test
Analyse et tests
Créer un plan de tests
Tests unitaires
Intégration conitnue
Interface
Analyse
Prototype
Implémentation
Déploiement
AWS (AmazonWebService)
Sentic
Recherche
Sentic
Application à la finance
Indicateurs
Stockage
Big data
Format générique pour les feeds
Système de récupération des données
Application stand-alone qui gère les "feed" pour le stockage sur BD distribuée
Les modules surlignés en vert sont ceux qui pourront être couverts dans le cadre de ce projet.

Portée


Phase 1
Yale UNIVERSITY
ÉTS
Le génie pour l'industrie
Transactions boursières
BD automate boursier
Ordres
Automate boursier PyAlgoTrade
Signal1: (achat, vente, attente, indicateur de contexte)
Signal2: (achat, vente, attente, indicateur de contexte)
Signal3: (achat, vente, attente, indicateur de contexte)
Signal3: (achat, vente, attente, indicateur de contexte)
Transactions
mess.
RabbitMQ
Phase 2
Apache Camel
Service orch. + persistence
RDF SPARQL
Ontologie boursière
APACHE MARMOTTA
NASTRAQ
WorldWideWeb
Content Crawler
Bloomberg
HDFS
Contenus multimédia à grande échelle
Extracteurs interconnectables
Orchestrateur d'extracteurs

Éléments bloquants:

Sébastien

- Début du réusinage du code: développement en Python non-maîtrisé.

Benjamin

- NULL

David

- NULL

Questions:

- Disponibilités Jeudi? (Sebastien)
- Imprimer les SCRUM au complet à chaque semaine?
- Le système permet-il d'acheter des PUT ? Vendre des CALL ?

Temps estimé: 4 heures individuelles. 12 heures total.

Éléments à faire:

- Réusinage du code (séparer le code du backtesting d'option de PyAlgoTrade). [10 heures]
- Diagramme pour illustrer l'architecture de la nouvelle solution. [3 heures]

La prochaine rencontre est prévue pour le **3 Février prochain**.

Scrum du vendredi, le 3 février 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Patrick Cardinal, ÉTS
    - Thomas Maketa, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS

Ce qui a été fait:

Benjamin: 6 heures

- Réflexion sur l'affichage global du site (présentation générale, graphiques, contenu)
- Clarifications des risques du projet
- Analyse du nouveau code.
- Recherche pour data options.

David: 3 heures

- Réflexion sur l'affichbage global du site (présentation générale, graphiques, contenu)
- Clarifications des risques du projet.
- Analyse du nouveau code.

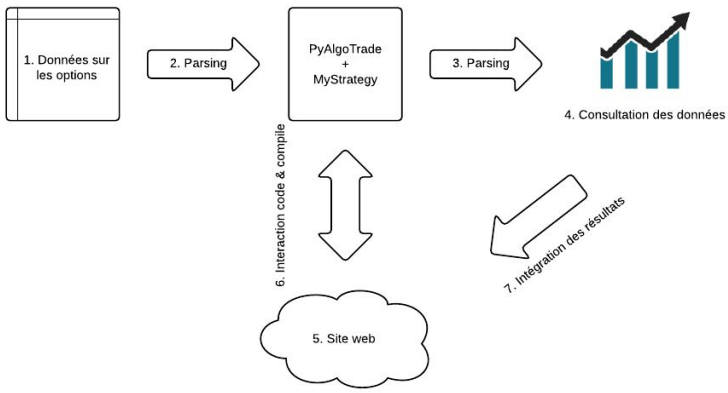
Sébastien: 3 heures

- Réflexion sur l'affichage global du site (présentation générale, graphiques, contenu)
- Clarifications des risques du projet.
- Analyse du nouveau code.
- Trouver une façon d'extraire les feeds du site Twitter.

→ Classe virtuelle avec Thomas pour répondre à nos questions sur les plans de tests et sur l'interface utilisateur à utiliser.

- Interface basé sur le site Quantopian

→ Diagramme d'ensemble de la solution PyAlgoTrade.



Éléments bloquants:

- Complexité du système

Questions:

- Quel data pour les options? Où les trouver?

Temps estimé:

- 8 heures à trois

Éléments à faire:

- Blitz de refactoring du code de PyAlgoTrade.
- Diagramme pour illustrer l'architecture de la nouvelle solution.
- Trouver une façon d'obtenir des données pour les options.
- Débuter le UI

Scrum du vendredi, le 10 février 2017

Rencontre:

- Lieu : Google Hangouts, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Thomas Maketa, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS

Ce qui a été fait:

Sébastien (10 heures)

- Refactoring de PyAlgoTrade.
- Tutoriels sur Django (Python MVC).
- Template de démo “AmCharts” et “Ace Editor”.
- Création de webapp Django et mockup html

David (12 heures):

- Refactoring de PyAlgoTrade
- Tutoriels Django
- Diagramme nouvelle architecture après refactoring.
- Création de webapp Django et mockup html

Benjamin (8 heures):

- Refactoring de PyAlgoTrade
- Tutoriels Django

Toutes les classes concernant les options ont été déplacées dans un nouveau package préfixé par **opt** à l’intérieur du package conteneur. Seulement deux packages originaux ont été affectés par le PFE précédent: **pyalgotrade.broker** et **pyalgotrade.strategy**. Il y a donc deux nouveaux packages: **pyalgotrade.broker.optbroker** et **pyalgotrade.strategy.optstrategy**. Ceci fait en sorte que les classes préexistantes de PyAlgoTrade ne sont pas touchées.

Diagramme de classes du package broker post-refactoring

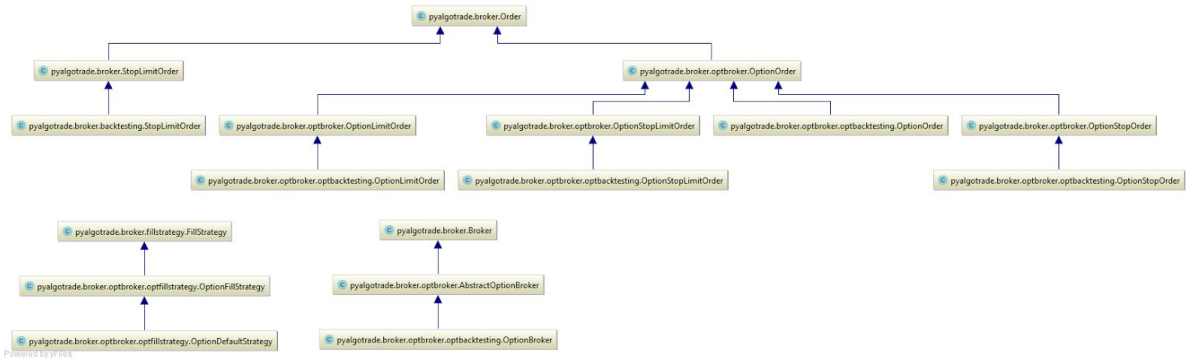
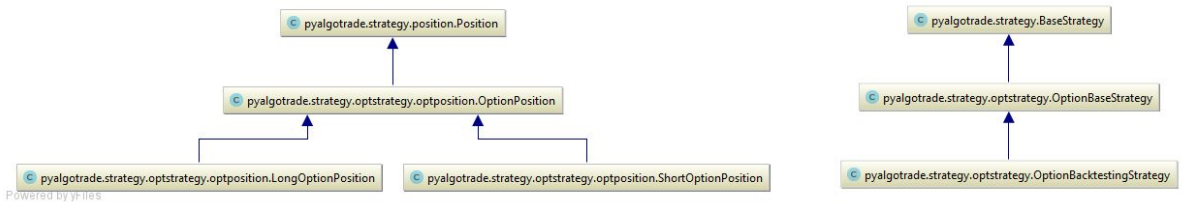
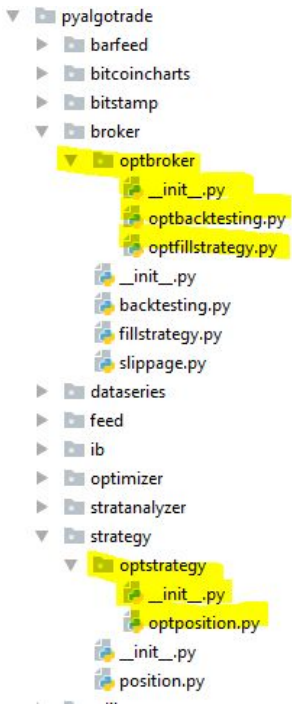


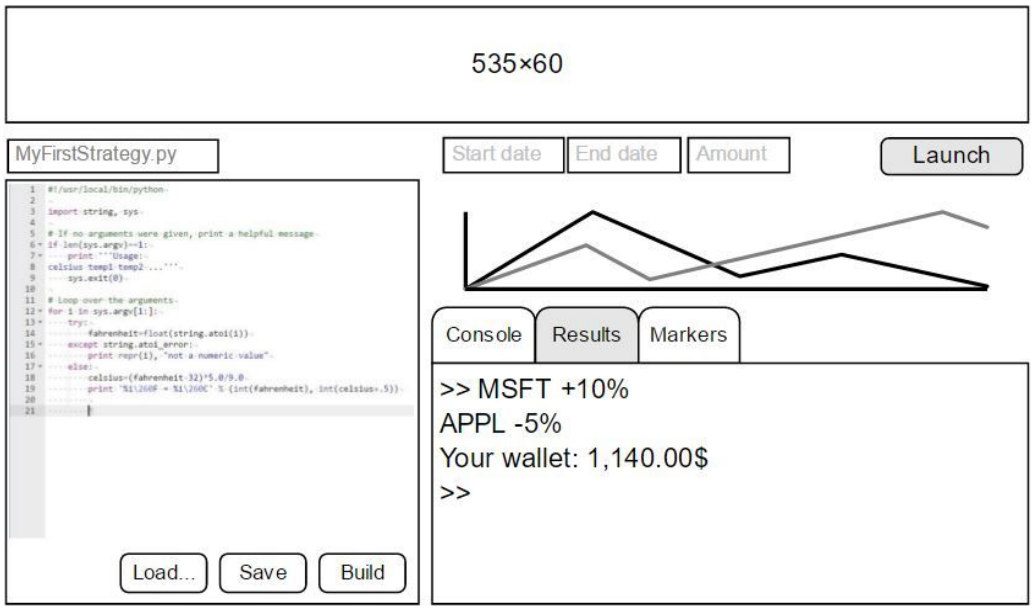
Diagramme de classes du package strategy post-refactoring



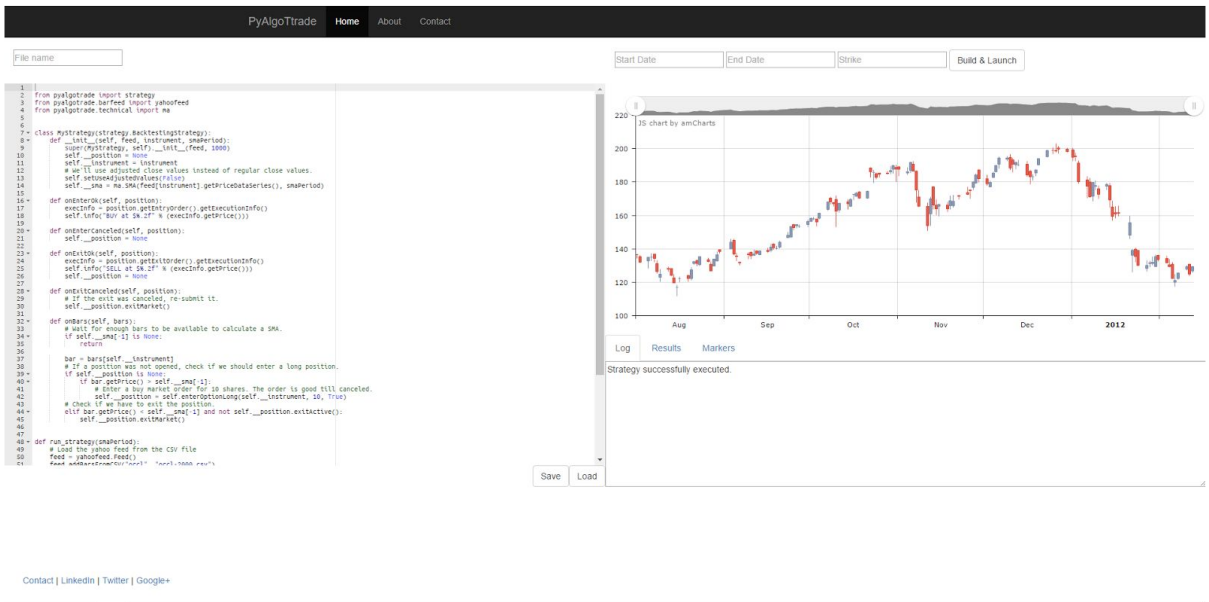
Arborescence PyAlgoTrade post-refactoring



Template HTML pour Ace Editor et les graphiques AmCharts



Site Django



Éléments bloquants:

Aucun.

Questions:

- Section SCRUMs hebdomadaires → Proposition de projet. (?)
- Portée de la compétition? Qu'est-ce que cela implique?
- Rapport final = Proposition de projet? Clarifier.
- Ajout de code pour mieux gérer les paramètres des options comme le strike price et la date d'expiration? Besoin de specs plus précis. Discussion sur le format des données en entrée? (Thomas) [Réglé]

Temps estimé:

- 9 heures chacun

Éléments à faire:

- Terminer les tutoriels sur Django MVC.
- Prendre strike price et expiration date du nom de fichier
- Rajouter des composants/inputs dans l'interface web actuelle (Instrument, PUT/CALL, Strike price, date d'expiration)
- Refactor de la méthode addBarsFromCSV(...) : prendre l'instrument uniquement en paramètres.
- Meeting sur 4 stratégies avec Alain et/ou Thomas (si on a les acces)

Scrum du vendredi, le 17 Février 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
  - Alain April, ÉTS
  - Thomas Maketa, ÉTS
  - Sébastien Santerre, ÉTS
  - Benjamin Lebois, ÉTS
  - David Méthot, ÉTS
- Rédacteur de SCRUM: Sébastien Santerre

Ce qui a été fait:

Collectif: (3 heures)

- Brainstorming et corrections de l'interface utilisateur web
- Analyse des possibilités de sources de données (API*) http://www.jarloo.com/google-stock-options-api/
- Analyse de la plateforme de compétition de trading https://contest.m-x.grid.inovestor.com/
-

Sébastien: (4 heures)

- Complétion des tutoriels en ligne Django MVC
- MAJ documentation
- Tentative de refactoring de généralisation de chargement d'instruments (~~hardcoding~~)

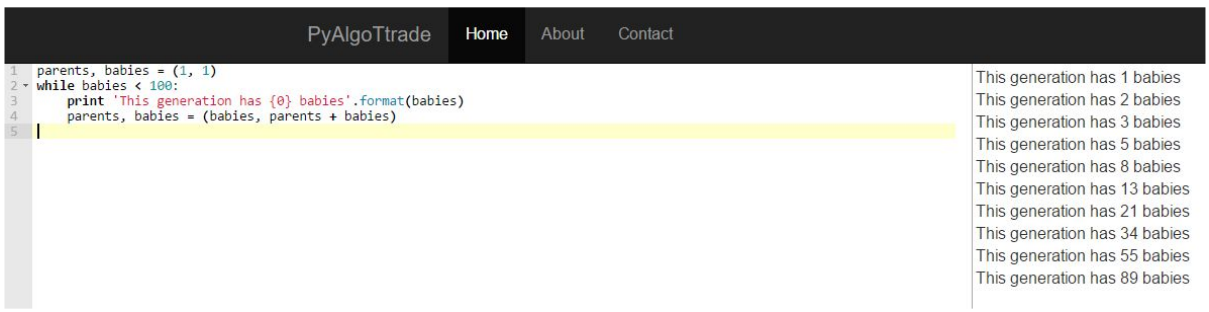
Benjamin: (6 heures)

- Complétion des tutoriels en ligne Django MVC
- Analyse d'utilisation de stratégie sur plusieurs instruments

David: (6 heures)

- Preuve de concept pour l'exécution dynamique du code python sur un serveur web Django.
- Analyse des tests unitaires existants dans la solution.

http://localhost:8000/testExec



Information intéressante à retrouver dans une classe “Instrument” qui décrit les détails d’un contrat :

Contract Description

AMAZON.COM INC

Security Type	OPT
Underlying	AMZN STK
Contract Month	FEB17
Expiration Date	FEB 17 '17
Last Trading Date	FEB 17 '17 16:00 EST
Strike	800
Right	Call
Currency	USD
Multiplier	100
Exchange	SMART
Trading Class	AMZN
Symbol	AMZN 170217C00800000
Sector	Communications
Industry	Internet
Category	E-Commerce/Products
Product Type	Equity
Settlement Method	Physical Delivery
Exercise Style	American

Trading Hours: February 17, 2017 Calendar

Total Available Hours 09:30 EST - 16:00 EST

Éléments bloquants:

- n/a

Questions:

- Source de données problématique. API?
- [!] IMPORTANT [!]
  Est-ce que PyAlgoTrade doit supporter écrire et exercer des options (PUT/CALL)?
- Explications sur une bonne stratégie sur les options.
- *Thomas: clarifications sur les exigences du produit à montrer lors de la présentation (attentes).

Plan d'actions:

- Mettre le UI en bonne et dû forme.
  Enlever les composants inutiles. Mettre une stratégie fonctionnelle hardcodé (démo).
  Exécuter une stratégie PyAlgoTrade à partir du site web Django.
- Créer la documentation pour chaque cas d’utilisation détaillé.
  Exemple: *L’utilisateur écrit une stratégie pour exercer un CALL.*
  - Contraintes logistiques
  - Source de données en entrées
  - Actions de l’utilisateur
  - Traitement des données (vision le plus approfondie possible)
  - Résultats (affichage et données)

La prochaine rencontre est prévue pour le **24 Février prochain**.

Scrum du vendredi, le 3 Mars 2017

****Jumelé à celui du 24 Février en raison des congés fériés.****
Rencontre:

- Lieu : Local A-3XXX, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Patrick Cardinal, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: Sébastien Santerre

Ce qui a été fait:
Collectif:

- Analyse de l'implémentation à faire en groupe

Sébastien: (20 heures):

- Les boutons "Load" et "Save" pour une stratégie sont fonctionnels.
- Refactoring du projet web pour enlever le conflit du namespace "pyalgotrade".
- Coding pour parser les paramètres du UI et récupérer les fichiers de données.
- Chargement des données du fichier dans le graphique de données
- Début rédaction de rapport d'étape

Benjamin: (20 heures)

- Déploiement d'un site web test sur DigitalOcean
- Début d'ajout des informations de contrat sur les "bars"
- Testcases pour les fonctionnalités d'options VS actions de PyAlgoTrade

David: (20 heures)

- Intégration du prototype d'exécution python sur la page d'index
- Exécution d'une stratégie côté serveur et retour du résultat.
- Recherche et analyse sur les possibilités de créer un "jail" Python pour empêcher l'exécution de code malveillant.
    - https://pypi.python.org/pypi/RestrictedPython
    - https://docs.python.org/2/library/rexec.html
    - UPDATE 1 Mars: Aucune des deux options est adéquate.
        - RestrictedPython ne permet pas l'écriture de classes
        - RExec est deprecated et inutilisable depuis Python 2.2
- Encapsulation de l'exécution de code dynamique Python dans un module RestrictedExecutionEnv.py

Classe RestrictedExecutionEnv

```
class RestrictedExecutionEnv:
    blacklist = {'import': 'import[ ]{1}[A-Za-z\.]*', 'open': '.*open[ |\(].*[ |\)].*'}

    def __init__(self):
        pass

    def executeUnstrustedCode(self, code):
        """Executes an untrusted block of code.
        In an effort to prevent malicious intent, before compilation, code is scanned to see if it contains any
        blacklisted elements.

        :param code: str
        :return: tuple(success: bool, msgList: str[])
        """
        messages = []
        blackListElementFound = False
        for keyword, regexp in self.blacklist.iteritems():
            if re.search(regexp, code):
                messages.append('Code cannot contain keyword: ' + keyword)
                blackListElementFound = True

        if blackListElementFound:
            return (False, messages)
        else:
            compiledCode = compile(code, '<string>', 'exec')
            exec (compiledCode, globals())
            return (True, ['Code successfully executed!'])
```

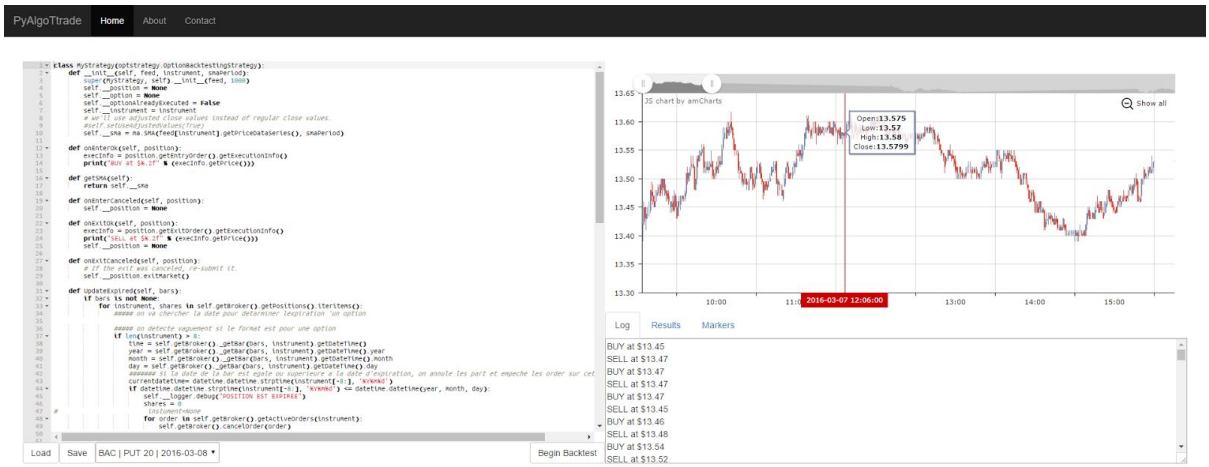
Charger plusieurs fichier csv

```
# Load the yahoo feed from the CSV file
feed = ibfeed.Feed()

parser = filenametool.Parser()

fileList = ["bac_20p20170217.csv","bac_21p20170217.csv","bac_22p20170217.csv"]

instru1 = parser.parse("bac_20p20170217.csv")
instru2 = parser.parse("bac_21p20170217.csv")
instru3 = parser.parse("bac_22p20170217.csv")

idList = []

feed.addBarsFromCSV(instru1.id, instru1.filename)
idList.append(instru1.id);
feed.addBarsFromCSV(instru2.id, instru2.filename)
idList.append(instru2.id);
feed.addBarsFromCSV(instru3.id, instru3.filename)
idList.append(instru3.id);

# Evaluate the strategy with the feed's bars.
myStrategy = MyStrategy(feed, idList)
myStrategy.run()
```

```python
class Parser(object):
    def parse(self, filename):
        slashIndex = filename.rfind('/')

        if (slashIndex > -1):
            filename = filename[slashIndex + 1:]

        zinstrument = filename[0:3]
        zStrikePrice = filename[4:6]
        zDate = filename[7:15]
        zID = filename[0:15]
        optiontype = filename[6]
        if (optiontype.lower() == "p"):
            optiontype = "PUT"
        elif (optiontype.lower() == "c"):
            optiontype = "CALL"
        else:
            optiontype = str(None)

        date = datetime.datetime.strptime(zDate, '%Y%m%d')

        instrument = ibfeed.Instrument(zinstrument,zStrikePrice,optiontype,date,filename,zID)

        return instrument
```

```python
#Option contract
class Instrument:
    def __init__(self, symbol, strike, right, expiry, filename, id):
        self.symbol = symbol
        self.strike = strike
        self.right = right
        self.expiry = expiry
        self.filename = filename
        self.id = id
```

```python
def execStrategy(request):
    if request.is_ajax() and request.POST:
        code = request.POST.get('strategy')
        with stdoutIO() as s:
            compiledCode = compile(code, '<string>', 'exec')
            exec(compiledCode, globals())

        execResult = s.getvalue()
        data = {'message': execResult}
        return HttpResponse(json.dumps(data), content_type='application/json')
    else:
        raise Http404
```



Éléments bloquants:

- n/a

Questions:

- Date de remise pour le rapport d'étape?
    - Clarifications (Contenu squelette)
- Date de remise pour le rapport final?
    - Date de présentation? Date de livraison du projet?
    - Clarifications (Contenu squelette)
- Quels sont les statistiques à afficher pour la performance de l'algorithme/stratégie?
    - Valeur de portefeuille initiale et finale
- Quels sont les éléments à afficher sur le graphique?
- Portée du projet

Plan d'actions:

- Stratégie de gestion des options peu exhaustive. (Séance avec Thomas à venir)
- Loading de données CSV: Permettre de loader plusieurs fichiers CSV et les lister dans le combobox pour choisir les éléments graphiques.
- Dans l'onglet "Results", afficher des statistiques de performance de l'algorithme
- Rédaction du rapport d'étape/final

La prochaine rencontre est prévue pour le **<u>10 Mars prochain</u>**.

Scrum du vendredi, le 10 Mars 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: Sébastien Santerre

Ce qui a été fait:

Sébastien: (8 heures):

- Loading de données CSV: Peupler le combobox avec un dossier de données.
- Affichage de statistiques dans l'onglet "Results" pour indiquer la performance. (proof of concept).
- Rédaction du rapport d'étape.

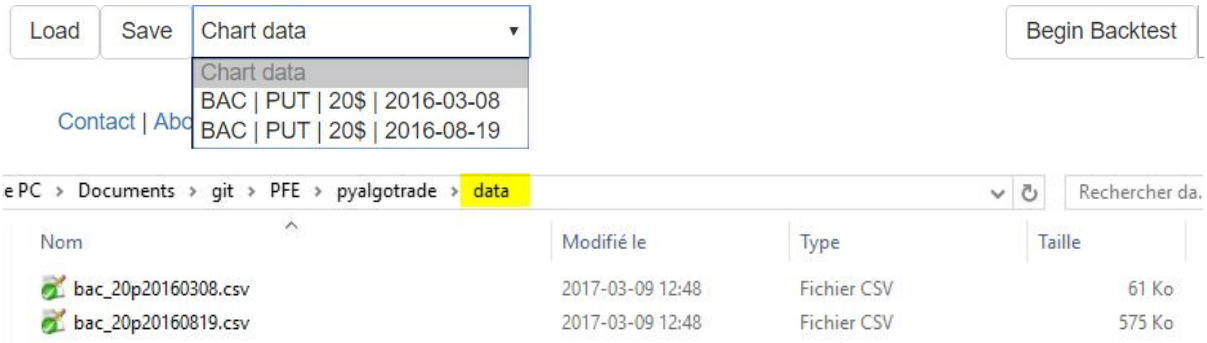
Benjamin: (8 heures)

- Rédaction du rapport d'étape
- Écrire classe de test et modifier le code pour fixer les problèmes que les test font apparaître.
- Étudier stratégie d'options

David: (8 heures)

- Apprendre sur Amazon EC2 et Elastic Beanstalk
- Configurer le déploiement du système
- Standardiser l'accès aux fichiers de données via une classe DataSourceHelper
- Rédaction du rapport d'étape

Combobox et Dossier "Data":



Classe DataSourceHelper:

```python
import os
from django.conf import settings

PYALGOTRADE_BASE = os.path.abspath(os.path.join(settings.BASE_DIR, ".."))
PYALGOTRADE_DATA_FOLDER = os.path.join(PYALGOTRADE_BASE, "data/")
PYALGOTRADE_SAMPLES_FOLDER = os.path.join(PYALGOTRADE_BASE, "samples/")


class DataSourceHelper:
    csvExtension = '.csv'

    def __init__(self):
        pass

    @staticmethod
    def getDataFilePath(csvFileName):
        """
        Builds a fully qualified path to a csv file in the dedicated data folder from a simple file name.
        :param csvFileName: str
        :return: Full path to file if it exists, empty path otherwise.
        """
        if csvFileName[-4:] != DataSourceHelper.csvExtension:
            csvFileName += DataSourceHelper.csvExtension
        fullPath = os.path.join(PYALGOTRADE_DATA_FOLDER, csvFileName)
        if os.path.isfile(fullPath):
            return fullPath
        else:
            # Try in samples?
            fullPath = os.path.join(PYALGOTRADE_SAMPLES_FOLDER, csvFileName)
            if os.path.isfile(fullPath):
                return fullPath
            else:
                return ''
```

Configuration AWS EBS:

```yaml
container_commands:
  collectstatic:
    command: "source /opt/python/run/venv/bin/activate && python webapp/manage.py collectstatic --noinput"

option_settings:
  "aws:elasticbeanstalk:application:environment":
    "PYTHONPATH": "/opt/python/current/app:$PYTHONPATH"
  "aws:elasticbeanstalk:container:python":
    WSGIPath: webapp/webapp/wsgi.py
  "aws:elasticbeanstalk:container:python:staticfiles":
    "/static/": "static/"
```

Lien d'exemple de stratégie d'options :

http://www.sciencedirect.com/science/article/pii/S1877050913011058
https://cmlviz.com/register/cml-trademachine-49-mo-limited-time-fb-promotion/

Éléments bloquants:

- Stratégie de gestion d'options peu exhaustive. Formation avec Thomas à suivre.

Questions:

- Section 4.b) Recommandations : Parle-t-on de la portée globale du projet ou de la

portée de nos objectifs initiaux? (Ben, Seb)

- Rapport section “Risques”: Est-ce qu’on garde le concept des phases dans nos risques? La phase 2 a été enlevé des objectifs du projet.
    -

Plan d'actions:

- Table des matières du rapport final
- Formation sur les stratégies de gestion d’options avec Thomas.
- Affichage des BUY/SELL sur le graphique en fonction du résultat de la stratégie.
- Afficher les statistiques de performance.

La prochaine rencontre est prévue pour le **17 Mars prochain**.

Scrum du vendredi, le 17 Mars 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Patrick Cardinal, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: David Méthot

Ce qui a été fait:

Collectif: (3 heures)

- Formation sur 4 stratégies de base en finances sur les options (Short/Long positions, Covered call, Butterfly, Bear Put Spread, Strangle)
- Finaliser le rapport d'étape

Sébastien: (8 heures):

- Ré-écriture du code pour le graphique AmCharts pour satisfaire les besoins du client. Comprend maintenant les BUY/SELL liés à une stratégie.

Benjamin: (8 heures)

- Tester stratégie sur platforme du concours et sur IB. Analyse du fonctionnement des écriture d'options dans une vrai platforme
- Finir test sur les nouvelles classes ajouté (backtesting)

David: (9 heures)

- Interconnecter PyAlgoTrade et l'application web via l'écriture du résultat de backtesting dans un fichier texte. La classe StrategyTransferObject s'occupe de la sérialisation de l'exécution d'une stratégie.
- Participer à l'ajout des BUY/SELL sur le graphique AmCharts.

Code du StrategyTransferObject:

```python
import datetime
import json
import os
from pyalgotrade.broker import Order

# Hardcoded paths to the temporary json transfer file to be used by webapp.
PYALGOTRADE_BASE = os.path.abspath(os.path.join(__file__, "../../../../"))
PYALGOTRADE_TEMP_DUMP_JSON_FILE = os.path.join(PYALGOTRADE_BASE, "data/temp/jsondump.json")


class StrategyTransferObject:
    BUY_ACTION_STR = "BUY"
    SELL_ACTION_STR = "SELL"

    def __init__(self):
        self.__initialEquity = 0
        self.__finalEquity = 0
        self.__buySellHistory = []

    def setInitialEquity(self, equity):
        """..."""
        self.__initialEquity = equity

    def setFinalEquity(self, equity):
        """..."""
        self.__finalEquity = equity

    def recordBuySell(self, action, instrument, date, price):
        """..."""
        buySell = ""
        if action == Order.Action.BUY:
            buySell = StrategyTransferObject.BUY_ACTION_STR
        elif action == Order.Action.SELL:
            buySell = StrategyTransferObject.SELL_ACTION_STR

        dateStr = date.strftime("%Y-%m-%d %H:%M:%S")
        buySell = {"buysell": buySell, "instrument": instrument, "date": dateStr, "price": price}
        self.__buySellHistory.append(buySell)

    def dumpToFile(self):
        """..."""
        jsonToDump = {"initialEquity": self.__initialEquity,
                      "finalEquity": self.__finalEquity,
                      "buySellHistory": self.__buySellHistory}
        with open(PYALGOTRADE_TEMP_DUMP_JSON_FILE, 'w') as jsonfp:
            json.dump(jsonToDump, jsonfp)
```

Surcharge des fonctions de base de la stratégie de backtesting d'options pour envoyer les données au StrategyTransferObject:

```
def onEnterOk(self, position):
    execInfo = position.getEntryOrder().getExecutionInfo()
    inst = position.getInstrument()
    action = pyalgotrade.broker.Order.Action.BUY
    self.__strategyTransferObject.recordBuySell(action, inst, execInfo.getDateTime(), execInfo.getPrice())

def onExitOk(self, position):
    execInfo = position.getExitOrder().getExecutionInfo()
    inst = position.getInstrument()
    action = pyalgotrade.broker.Order.Action.SELL
    self.__strategyTransferObject.recordBuySell(action, inst, execInfo.getDateTime(), execInfo.getPrice())

def onStart(self):
    self.__strategyTransferObject.setInitialEquity(self.getBroker().getEquity())

def onFinish(self, bars):
    self.__strategyTransferObject.setFinalEquity(self.getBroker().getEquity())
    self.__strategyTransferObject.dumpToFile()
```

Affichage des BUY/SELL sur le graphique AmCharts:

Éléments bloquants:

- n/a

Questions:

- n/a

Plan d'actions:

- Approbation table des matières du rapport final.
- Rédaction du rapport final.
- Ajustement des BUY/SELL pour le AmCharts.
  Texte d'affichage ++ Afficher seulement les BUY/SELL sur les portions zoomées
- Trouver une façon de comparer notre système VS un autre système comparable
- Remplir les spécifications pour montrer que notre système peut supporter les 4 statégies présentées par Thomas lors de sa présentation.
- Approbation de Thomas pour l'outil développé jusqu'à présent.

La prochaine rencontre est prévue pour le **24 Mars prochain**.

Scrum du vendredi, le 24 Mars 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: David Méthot

Ce qui a été fait:

Sébastien: (8 heures):

- Correctifs apportés au graphique AmCharts pour le formatage de date sur l'étiquette de l'axe des X.
- Travail sur le rapport final.

Benjamin: (8 heures)

- Finir plan de test et test sur les nouvelles fonctionnalités et les classes de stratégies.
- Refactor de la classe filename.
- Travail sur le rapport final, section sur les tests.

David: (9 heures)

- Affichage dynamique des BUY/SELL selon le niveau de zoom.
- Travail sur le rapport final.

Éléments bloquants:

- n/a

Questions:

- n/a

Plan d'actions:

- Poursuite des travaux sur le rapport final.
- Rédiger les spécifications pour les modifications à apporter à PyAlgoTrade pour être en mesure d'implémenter les 4 stratégies de base. (Dans la section futur du projet / recommandations du rapport final)
- Préparation d'un document de présentation final.
- Évaluer la complexité de l'intégration de la gestion de l'expiration (++ écriture d'achat/vente d'actions).

La prochaine rencontre est prévue pour le **31 Mars prochain**.

Scrum du vendredi, le 31 Mars 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Thomas Maketa, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: Sébastien Santerre

Ce qui a été fait:

Sébastien: (8 heures):

- Rédaction rapport final.

Benjamin: (6 heures)

- Rapport final
- Vérifier comment ajouter les options dans PyAlgoTrade

David: (8 heures)

- Rédaction rapport final.
- Analyse de code pour gérer l'expiration.

Éléments bloquants:

- n/a

Questions:

- Date de remise pour le rapport d'étape?
- Date de présentation?

Plan d'actions:

- Poursuite des travaux sur le rapport final.
- Poursuite des travaux sur la présentation.

Scrum du vendredi, le 7 Avril 2017

Rencontre:

- Lieu : ÉTS, 12h00
- Personnes présentes:
    - Alain April, ÉTS
    - Thomas Maketa, ÉTS
    - Sébastien Santerre, ÉTS
    - Benjamin Lebois, ÉTS
    - David Méthot, ÉTS
- Rédacteur de SCRUM: Sébastien Santerre

Ce qui a été fait:

Sébastien: (11 heures):

- Rapport final
- Présentation finale
- Trouver une façon de récupérer les données de TickSmith en suivant le protocole fourni par Thomas.

Benjamin: (10 heures)

- Rapport final
- Présentation finale

David: (10 heures)

- Rapport final
- Présentation orale du pfe
- Regardé les données de l'api TickSmith → Le format de sortie est étrange, revoir...

Éléments bloquants:

- n/a

Questions:

- n/a

Plan d'actions:

- Présentation finale

La prochaine rencontre est prévue pour le **<u>11 Avril 2017 à 13h00</u>**.