

Mastère expert développement logiciel

Seddik Mehdi – Ynov 2025

**Livrable Bloc 2**

CONCEVOIR ET DÉVELOPPER DES APPLICATIONS LOGICIELLES

Seddik Mehdi

Table des matières

[Introduction 3](#_Toc206528355)

[1 – Architecture Logicielle 3](#_Toc206528356)

[Technologies et frameworks 3](#_Toc206528357)

[Choix de l’architecture 4](#_Toc206528358)

[2 – Cahier de recettes 5](#_Toc206528359)

[Cahier de recette 5](#_Toc206528360)

[Suivi des bugs avec github issues, 6](#_Toc206528361)

[3 – Cycle de développement 6](#_Toc206528362)

[Environnement de developpement 6](#_Toc206528363)

[Protocole de déploiement continue 6](#_Toc206528364)

[Protocole d’intégration continue 7](#_Toc206528365)

[Sécurité 7](#_Toc206528366)

[Accessibilité 7](#_Toc206528367)

[4 – Documentations 7](#_Toc206528368)

[Guide pour la feature Dashboard 9](#_Toc206528369)

[Annexes 11](#_Toc206528370)

# Introduction

L'objectif principal de ce projet est de concevoir et développer une plateforme qui permet aux « quants » (abréviation pour « quantitatives », des professionnels responsables de la prise de décision financière dans les institutions de finance, notamment via des formules mathématiques complexe et des algorithmes informatique) de visualiser l’état des marchés, et de gérer efficacement leurs algorithmes de trading automatisés. La plateforme Delta répond à deux besoins majeurs de la communauté des traders : l'affichage de graphiques financiers, ainsi que la gestion complète des algorithmes, incluant le déploiement, le logging et le démarrage/arrêt.

Cette plateforme est conçue pour automatiser les tâches DevOps et de gestion des algos en production, permettant ainsi aux quants de se concentrer sur le développement et l'amélioration des algorithmes. Le but ultime de ce projet est de faciliter la collaboration entre les développeurs, les quants et les opérateurs pour optimiser les performances des algorithmes de trading.

Dans ce dossier, nous présentons la phase de développement du projet.

1 – Architecture Logicielle

## Technologies et frameworks

Pour développer ce projet, nous avons choisi une approche polyvalente qui combine les avantages de différentes technologies. Nous sommes particulièrement fiers de notre choix pour les technologies suivantes :

Le frontend est construit avec React et TypeScript. Nous avons opté pour ces technologies car elles offrent une excellente combinaison de performances, de flexibilité et de facilité d'utilisation. La bibliothèque de composants React Shadcn nous permet également un design UX/UI plus raffiné et une meilleure gestion des interfaces utilisateur complexes.

Le backend est écrit en Golang, un language qui a gagné en popularité ces dernières années en raison de sa vitesse, de son efficacité et de sa facilité d'utilisation. Nous avons choisi Golang pour son écosystème riche, ses bibliothèques puissantes et sa communauté active. Cette langue est également particulièrement adaptée au développement cloud native, car les programmes écrits en GO sont très faciles à containeriser, et la majorité des outils cloud sont écrits avec ce langage (par exemple Kubernetes, Docker, etc...).

Pour stocker les logs des algorithmes, nous avons choisis la base de données PostgreSQL, nous aurions pu utiliser une base de donnée Timeseries étant donné que nous n’avons pas de donnée relationnelle, mais étant donné que PostgreSQL permet les deux, il nous a semblé judicieux de partir dessus si jamais nous avons besoin d’étendre les fonctionnalités de l’application.

Pour permettre une communication fiable entre le backend et les algorithmes, nous avons choisi le protocole gRPC. Cette technologie offre une haute performance (plus rapide que les interfaces REST, car elle utilise un protocole qui envoie des données compressées en binaire).

Pour ce projet, nous avons fait le choix de déployer l'application dans un cluster Kubernetes pour gérer les conteneurs et assurer leur déploiement automatisé. Cette technologie offre une excellente scalabilité et permet d'automatiser une multitude de tâches telles que le déploiement, le scaling horizontal, la gestion des ressources et la surveillance de l'application.

## Choix de l’architecture

Notre projet est basé sur une architecture micro-services, qui consiste en un ensemble d'unités fonctionnelles indépendantes, chacune ayant une responsabilité spécifique. Dans notre cas, le backend est composé d'un service maitre et de plusieurs services esclaves, chacun représentant un algorithme différent. (Voir l’architecture complète en Annexe 1.)

Le service maitre reçoit les requêtes du frontend et les dispatches sur les algorithmes concernés. Ce service est responsable de la gestion des requêtes entrantes et de la coordination entre les différents services esclaves. Il s'assure que les données nécessaires sont disponibles pour chaque algorithme et qu'ils soient exécutés de manière efficace.

Les services esclaves, quant à eux, représentent chaque algorithme individuel. Ils sont responsables de l'exécution spécifique du traitement des requêtes, en fonction de leur domaine d'expertise. Chaque service exclave est containerisé pour assurer une isolation et une sécurité maximales. Le développement de ces algorithmes sont sous la responsabilité des Quants, nous fournirons uniquement des programmes mocké en guise d’exemple.

Le déploiement et la gestion des services sont gérées par un cluster Kubernetes, qui permet de déployer les conteneurs automatiquement et de gérer les ressources en temps réel. Cette architecture nous permet de répondre rapidement aux besoins évoluant du projet et d'améliorer l'efficacité globale de l'application.

Dans ce modèle, chaque service maitre et esclave est conçu pour fonctionner de manière autonome, mais également pour interagir avec les autres services pour atteindre le résultat souhaité. Cette architecture nous permet d'améliorer la flexibilité, la scalabilité et la résilience globale de l'application.

L'utilisation d'une architecture micro-services offre plusieurs avantages, notamment :

* Flexibilité : chaque service est indépendant et peut être mis à jour ou modifié sans affecter les autres services, c’est très pratique pour la mise à jour d’un algorithme sans devoir redéployer l’application complète.
* Scalabilité : les conteneurs permettent une gestion efficace des ressources et une évolution de la charge de travail, si par exemple un algorithme est surchargé, on peut facilement le dupliquer, voir créer un réplica sur une autre machine du cluster.
* Résilience : si un service est indisponible, les autres peuvent prendre son relais pour continuer à fonctionner.

# 2 – Cahier de recettes

## Cahier de recette

Le cahier de recette permet de valider la qualité et la conformité du projet livré au client, a chaque tests effectué, un commentaire est rédigé par l’équipe QA qui valide ou invalide la fonctionnalité.  
Les bugs et axes d’amélioration font l’objet d’un ticket assigné à l’équipe technique, vous pouvez en trouver un exemple en Annexe 3

|  |  |  |  |
| --- | --- | --- | --- |
| Fonctionnalité | Test effectué | Résultat attendu | Commentaire |
| Dashboard |  | | |
|  | **Chargement rapide du Dashboard** | Le Dashboard charge en moins de 3 secondes | OK |
|  | **Ouvrir un graphique.** | Un graphique s’affiche sur la page. | OK |
|  | **Supprimer un graphique.** | Un graphique disparait. | OK |
|  | **Glisser/déposer un graphique.** | Le graphique suit le curseur. | OK |
|  | **Sélectionner une action.** | Le graphique charge le graphique de cette action. | OK |
|  | **Changer la timeframe.** | Le graphique recharge avec des bougies adapté à la timeframe. | OK |
|  | **Sauvegarder le Dashboard.** | L’utilisateur attribue un nom au Dashboard, qu’il pourra ré-ouvrir plus tard. | Non-implémenté |
|  | **Charger un Dashboard sauvegardé.** | L’utilisateur charge un Dashboard et les graphiques de ce Dashboard apparaissent a l’écran. | En cours de développement |
| Algorithmes |  | | |
|  | **Afficher la liste des algorithmes déployé** | Un tableau qui liste les algorithmes apparait avec leurs noms, leurs balances, et leurs état | OK |
|  | **Discovery des algorithmes sur le réseau** | En cliquant sur un bouton, l’application peut découvrir automatiquement des nouveaux algorithmes déployés sur le reseau | Non fonctionnel **depuis le frontend** à ce jour |
|  | **Activer un algorithme** | En activant un algorithme, son état passe en « Running », et sa balance se met a jour en temps réel | Non fonctionnel **depuis le frontend** à ce jour |
|  | **Désactiver un algorithme** | En désactivant un algorithme, son état passe en « Stopped » | Non fonctionnel **depuis le frontend** à ce jour |
|  | **Consulter les logs d’un algorithme** | Un menu permet de visualiser l’historique des transactions | En cours de développement |
| Autre |  | | |
|  | **Rechercher une page ou une section sur l’application entière** | La barre de recherche permet de naviguer facilement entre les pages et sections | OK |
|  | **Changer de thème** | Cliquer sur l’icône du thème change le mode d’affichage (dark theme ou light theme) | OK |
|  | **S’authentifier avec une empreinte digitale** | A l’ouverture de l’application, la page d’authentification bloque l’accès et demande une vérification par biométrie | En cours de développement |

## Suivi des bugs avec github issues,

# 3 – Cycle de développement

## Environnement de developpement

Pour créer un environnement de développement fonctionnel, nous utilisons une combinaison d'outils essentiels :   
En premier lieu, Git et GitHub pour gérer versionner et collaborer avec d'autres développeurs.   
Comme éditeur de code nous utilisons principalement VSCode, mais certaines parties ont été écrite sur Neovim (il s’agit d’un éditeur de texte terminal beaucoup plus performant et amusant que VSCode)

Pour gérer les dépendances nous avons choisi Yarn comme gestionnaire de packages, connu pour etre une excellente alternative a npm notemment pour sa rapidité. Pour executer les tests unitaire du frontend nous avons utilisé Jest, et pour le backend, le langage go a déjà un système de test builtin.  
Pour lancer certains services sans avoir à les compiler nous utilisons Docker, ce qui nous permet via la commande Docker compose de démarrer plusieurs services avec leurs dépendances nécessaires en quelques clics.

## Protocole de déploiement continue

Au fur et à mesure que l’équipe de développement livre de nouvelles fonctionnalités, l’adoption d’une culture **DevOps** devient essentielle afin de garantir la fluidité et la fiabilité du cycle de développement. Le DevOps désigne un ensemble de pratiques visant à rapprocher les équipes de développement et d’exploitation, en automatisant et en intégrant les processus de livraison logicielle. Dans ce projet, un protocole de **déploiement continu (CD)** a été mis en place, permettant de maintenir le serveur de production régulièrement à jour avec la dernière version validée du logiciel. Pour ce faire, nous avons défini un système de versionning suivant le schéma MAJEUR.MINOR.PATCH et une stratégie de déploiement sur deux environnements distincts : préproduction et production.

Chaque Pull Request ouverte sur la branche principale déclenche automatiquement le déploiement d’une **release candidate** en préproduction, rendant ainsi les nouvelles fonctionnalités accessibles pour l’équipe QA, qui les valide une à une. Lorsqu’une version est jugée prête pour la mise en production, un tag de release est créé sur GitHub, ce qui déclenche le déploiement automatique de l’application en production. Afin d’assurer la traçabilité et la possibilité de rollback en cas de problème, les images Docker sont historisées sur le registre GitHub Container Registry (Voir en annexe 2).

L’infrastructure repose sur une approche de containerisation via Docker, orchestrée au sein d’un cluster Kubernetes déployé sur un homelab, une machine personnelle servant de serveur. Cette architecture permet d’expérimenter et de mettre en place un pipeline complet, allant de l’intégration continue jusqu’au déploiement continu, tout en garantissant un haut niveau de reproductibilité et de fiabilité.

## Protocole d’intégration continue

En parallèle du déploiement continu, une stratégie **d’intégration continue (CI)** a été mise en place afin de garantir la qualité du code livré. Chaque **Pull Request** déclenche automatiquement une série de tests automatisés permettant de détecter d’éventuelles régressions avant la fusion dans la branche principale. De plus, un processus de vérification de la qualité du code est intégré via **ESLint** pour l’analyse statique et **Prettier** pour l’uniformisation du formatage. Ces outils, également configurés comme extensions dans Visual Studio Code, assurent une cohérence de style et une meilleure lisibilité du code. Enfin, ces contrôles sont validés au sein d’une **pipeline d’intégration continue**, qui centralise les résultats et bloque la fusion si les tests ou les règles de linting échouent, garantissant ainsi la robustesse et la maintenabilité du projet.

Une image contenant texte, logiciel, Police, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

## Sécurité

La dimension sécurité a été intégrée dès la phase de conception afin de protéger les données sensibles et de garantir la fiabilité, la confidentialité et l’intégrité de l’application. Conformément aux bonnes pratiques de développement sécurisé, plusieurs mesures techniques ont été mises en œuvre :

Gestion des secrets : Les mots de passe, clés API et autres informations sensibles sont stockées dans GitHub Secrets, évitant toute exposition dans le code source ou les commits. Cette pratique respecte les recommandations de sécurité des infrastructures cloud et limite le risque de fuite accidentelle.

Authentification biométrique via WebAuthn : Une fonctionnalité d’authentification renforcée a été développée, permettant aux utilisateurs de se connecter à l’application grâce à leur empreinte digitale. L’intégration de WebAuthn (FIDO2) assure un niveau élevé de sécurité en remplaçant les mots de passe traditionnels par une authentification sans mot de passe, résistante aux attaques par phishing et aux compromissions de base de données.

Mise à jour automatisée des dépendances : L’outil **Dependabot** de GitHub est activé pour surveiller en continu les bibliothèques tierces utilisées dans le projet. En cas de vulnérabilité critique détectée (ex : CVE), un pull request automatique est généré, alertant l’équipe pour une correction rapide. Ce processus contribue à maintenir l’application à jour et sécurisée selon les principes de DevSecOps.

Infrastructure sécurisée en production : Le déploiement en production repose sur un domaine personnalisé (DNS) acquis via un service de nom de domaine (CloudFlare). Un certificat SSL/TLS (via Let’s Encrypt) a été mis en place pour chiffrer toutes les communications entre l’utilisateur et le serveur, garantissant un accès sécurisé via HTTPS. Cette mise en œuvre répond aux exigences de conformité web (RGAA, GDPR) et aux bonnes pratiques de sécurité réseau.

Conformité aux 10 failles OWASP : Le code a été conçu pour minimiser les risques liés aux principales vulnérabilités décrites par l’OWASP (ex : injection SQL, XSS, CSRF). Des contrôles de validation, d’encodage et d’autorisation sont implémentés au niveau des endpoints.

## Accessibilité

# 4 – Documentations

Pour le manuel d’installation et de déploiement, vous pouvez également les retrouver dans le repo github, ils sont rédigé en Markdown sur le Readme,

Pour le manuel d’utilisation, vous pouvez retrouver ici un guide pour une des features clé du logiciel :

### Guide pour la feature Dashboard

Dans ce guide vous allez apprendre à créer votre premier Dashboard

1. Ouvrez votre navigateur web et naviguez sur le lien qui vous a été fournis lors du déploiement initial de la solution
2. La page de connexion suivante se présente à vous, renseignez votre nom d’utilisateurUne image contenant texte, capture d’écran, Police, conception

   Le contenu généré par l’IA peut être incorrect.   
   Puis cliquez sur « login »
3. Un popup va s’ouvrir et vous demander de placer votre doigt sur votre lecteur d’empreinte  
   Une image contenant texte, capture d’écran, logiciel, conception

   Le contenu généré par l’IA peut être incorrect.
4. Une fois connecté, naviguez sur la page Dashboard, pour ce faire deux options s’offrent à vous :  
   Vous pouvez soit cliquer sur la barre de recherche et taper « Dashboard », puis entrerUne image contenant texte, capture d’écran, logiciel, Logiciel multimédia

   Le contenu généré par l’IA peut être incorrect.  
   Soit ouvrir le menu à gauche, puis cliquer sur le bouton «  Dashboard »

Une image contenant texte, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.

1. Vous allez atterrir sur la page Dashboard, pour créer un graphique, faite un clic droit sur le fond puis sélectionnez « New Panel » Une image contenant capture d’écran, texte, Police, Graphique

   Le contenu généré par l’IA peut être incorrect.
2. Un graphique s’ouvre et vous pouvez le configurer via les deux options en haut du panel :   
   Une image contenant capture d’écran, texte, diagramme

   Le contenu généré par l’IA peut être incorrect.

Annexes

Une image contenant capture d’écran, diagramme

Le contenu généré par l’IA peut être incorrect.

Figure 1: Schema haut niveau de l'architecture logicielle

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

Figure 2: Image docker hébergé sur le github container registry

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 3: Exemple de ticket bug