Cahier des Charges : Système de Trading Algorithmique Avancé

1. Introduction

Ce cahier des charges définit les spécifications pour le développement d'un système de trading algorithmique avancé. L'objectif est de créer une plateforme modulaire, robuste et performante qui automatise l'exécution de stratégies de trading tout en intégrant une gestion stricte des risques, des mécanismes d'optimisation et un orchestrateur central pour coordonner l'ensemble du processus.

2. Objectifs du Projet

- **Automatisation** : Exécuter des stratégies de trading de manière 100% automatique, 24h/24 et 7j/7, en supprimant l'émotion et les délais humains[reference:0].
- **Gestion du Risque** : Intégrer en profondeur un système de gestion des risques pour protéger le capital et limiter les pertes, élément fondamental de tout robot de trading efficace[reference:1].
- **Optimisation** : Permettre le réglage fin et l'optimisation continue des paramètres des stratégies pour s'adapter aux conditions de marché changeantes.
- **Orchestration** : Coordonner l'exécution simultanée de multiples stratégies sur différents actifs et timeframes via un moteur centralisé.
- **Surveillance & Reporting** : Fournir une visibilité en temps réel sur les performances, les positions ouvertes et l'état du système.

3. Portée du Projet

Le projet couvre le développement d'un logiciel backend (orchestrateur, moteurs de stratégie, gestion des risques) et d'une interface de surveillance/dashboard. Il n'inclut pas le développement d'interfaces client de trading grand public (type MetaTrader), mais se connectera aux APIs des courtiers/exchanges.

4. Exigences Fonctionnelles Détaillées

4.1. Stratégies de Trading (Moteur de Stratégie)

- **Modularité** : Chaque stratégie doit être une classe indépendante, héritant d'une classe de base commune (`BaseStrategy`). L'ajout d'une nouvelle stratégie ne doit pas nécessiter de modification du code de l'orchestrateur[reference:2].
- **Configuration par Fichier** : Les stratégies doivent être paramétrées via des fichiers de configuration (ex: YAML) définissant le symbole, le timeframe, les paramètres techniques et les règles de risque[reference:3].
- **Génération de Signaux** : Chaque stratégie doit analyser les données de marché (candles) et produire des signaux de trading clairs (`BUY`, `SELL`, `HOLD`).

4.2. Système de Gestion des Risques Intégré

Le robot doit intégrer des mécanismes de protection du capital, condition sine qua non d'une gestion saine[reference:4].

- **Money Management** :
  - Limite du risque par trade (ex: 1% du capital)[reference:5].
  - Calcul dynamique de la taille de position basé sur la volatilité et le capital disponible.
- **Protections par Trade** :
  - Ordres `Stop-Loss` (fixe ou trailing).
  - Ordres `Take-Profit`.
  - Délai minimum entre les trades.
- **Protections Globales du Portefeuille** :
  - Drawdown maximum quotidien et global (ex: 1% journalier, 10% total)[reference:6].
  - Exposition maximum par actif ou secteur.
  - Coupe-circuit (circuit breaker) arrêtant toute activité en cas de pertes anormales.
- **Gestion des Risques Techniques** : Mécanismes pour gérer les pannes de connexion, les délais d'API et les données erronées[reference:7].

4.3. Systèmes d'Optimisation

L'optimisation est un processus continu pour adapter le robot aux conditions de marché[reference:8].

- **Backtesting** : Outil intégré pour tester les stratégies sur des données historiques, avec des métriques de performance (profit factor, drawdown, ratio de Sharpe)[reference:9].
- **Forward Testing (Paper Trading)** : Exécution en temps réel sur un compte démo pour valider les performances avant le déploiement en live[reference:10].
- **Optimisation des Paramètres** :
    - Recherche par grille ou algorithmes (ex: Bayesian Optimization) pour trouver les meilleurs paramètres sur une période historique.
    - **Garder-fou contre la sur-optimisation** : Validation des paramètres optimisés sur une période historique "hors échantillon" (out-of-sample) pour éviter un surajustement inefficace en conditions réelles[reference:11][reference:12].

4.4. Orchestrateur Central (Strategy Engine)

L'orchestrateur est le cœur du système, responsable de la coordination de tous les composants[reference:13].

- **Gestion du Cycle de Vie** : Chargement, initialisation, démarrage, mise en pause et arrêt des stratégies à partir de leur configuration[reference:14].
- **Gestion des Données** : Acquisition efficace des données de marché depuis diverses sources (APIs), avec mise en cache et diffusion aux stratégies concernées pour éviter les requêtes redondantes[reference:15].
- **Coordination des Signaux** : Collecte des signaux de toutes les stratégies actives, application des règles de risque globales, et transmission des ordres validés au module d'exécution.
- **Suivi des Statistiques** : Enregistrement en temps réel des performances (signaux générés, trades, P&L) par stratégie[reference:16].

4.5. Module d'Exécution des Ordres

- **Connecteurs Multiples** : Support pour plusieurs courtiers ou exchanges (ex: Binance, Kraken, MetaTrader via API).
- **Gestion des Ordres** : Envoi, annulation, suivi de l'état des ordres (filled, partial, rejected).
- **Slippage et Liquidité** : Logique pour gérer l'exécution dans des marchés peu liquides.

4.6. Dashboard de Surveillance & Alertes

- **Vue Globale** : Tableau de bord affichant le P&L global, le drawdown, l'exposition et l'état des stratégies.
- **Détail par Stratégie** : Graphiques de performance, historique des trades, métriques clés.
- **Journal des Événements** : Logs système, signaux générés, exécutions d'ordres et erreurs.
- **Système d'Alerte** : Alertes configurables (email, Telegram) pour les évén critiques (drawdown atteint, stratégie inactive, erreur d'API).

5. Exigences Non-Fonctionnelles

- **Performance** : Latence minimale entre la génération du signal et l'envoi de l'ordre (< 1 seconde). Capacité à gérer des données de plusieurs paires/marchés simultanément.
- **Fiabilité & Disponibilité** : Taux de disponibilité cible > 99.5%. Mécanismes de reprise sur erreur (retry, fallback).
- **Sécurité** : Stockage sécurisé des clés API (chiffrement). Audit des logs. Authentification pour l'accès au dashboard.
- **Maintenabilité & Évolutivité** : Code modulaire et bien documenté. Facilité d'ajout de nouvelles stratégies, connecteurs ou règles de risque.
- **Facilité d'Utilisation** : Interface de configuration et de surveillance claire, même pour des utilisateurs non-développeurs.

6. Architecture Technique (Proposition)

- **Langage** : Python (pour sa richesse en bibliothèques financières et de data science : `pandas`, `numpy`, `ccxt`, `backtrader/vectorbt`).
- **Orchestrateur** : Service principal (`StrategyEngine`) gérant les boucles de travail, la configuration et la coordination[reference:17].
- **Base de Données** : Base de données relationnelle (PostgreSQL) ou Time-Series (InfluxDB) pour stocker les données de marché, les configurations, les signaux et l'historique des trades.
- **Message Queue** (Optionnel) : Utilisation de Redis ou RabbitMQ pour découpler la génération de signaux et l'exécution des ordres, améliorant l'évolutivité.
- **Dashboard** : Application web développée avec un framework léger (FastAPI + React, ou Streamlit/Dash pour un prototype rapide).

- **Déploiement** : Conteneurisation avec Docker. Orchestration possible avec Docker Compose ou Kubernetes pour les environnements de production.

7. Livrables

1. **Code Source** : Dépôt Git contenant tout le code de l'application.
2. **Documentation Technique** : Guide d'architecture, d'installation, de configuration et d'API.
3. **Documentation Utilisateur** : Guide pour configurer les stratégies, interpréter le dashboard et gérer les incidents.
4. **Scripts de Déploiement** : Fichiers Dockerfile et docker-compose.yml.
5. **Environnements de Test** : Configuration pour le backtesting, le forward testing et la production.

8. Calendrier (Estimation)

- **Phase 1 (2-3 mois)** : Architecture de base, module de données, classe `BaseStrategy`, et 1-2 stratégies simples avec backtesting.
- **Phase 2 (2-3 mois)** : Développement de l'orchestrateur, du système de gestion des risques de base et du connecteur d'exécution pour un exchange.
- **Phase 3 (1-2 mois)** : Développement du dashboard de surveillance et des systèmes d'optimisation avancée.
- **Phase 4 (1 mois)** : Tests intensifs (backtest, forward test), correction de bugs, finalisation de la documentation et déploiement pilote.

9. Conclusion

Ce cahier des charges décrit les fondations d'un système de trading algorithmique professionnel, alliant automatisation et contrôle rigoureux. La priorité absolue doit être placée sur la **gestion des risques** et la **fiabilité** du système, les performances et l'optimisation venant ensuite. Une approche modulaire et itérative, en validant chaque composant par des tests rigoureux, est essentielle pour réussir ce projet complexe.