Rapport de Projet - Crypto Follow

1. Présentation du Projet

Crypto Follow est une plateforme complète de surveillance et d'analyse des marchés de cryptomonnaies. Elle combine une application web interactive pour l'analyse et la simulation de trading avec un système backend robuste pour la collecte de données en temps réel et la gestion des alertes.

Fonctionnalités Clés Implémentées

- **Dashboard Interactif** : Visualisation des prix, graphiques en chandeliers, et indicateurs techniques (SMA), avec des filtres temporels (1J, 7J, 30J, 90J).
- **Système d'Alertes** : Configuration d'alertes de prix personnalisées (supérieur/inférieur).
- **Notifications** : Intégration de Webhooks Discord pour les notifications en temps réel.
- **Portefeuille Virtuel** : Simulation de trading avec un capital de départ de 10 000 $, calcul des P&L en temps réel.
- **Prédictions** : Module de prévision basé sur la régression linéaire pour estimer les tendances futures.
- **Authentification** : Système sécurisé via Supabase Auth (Email/Password) avec possibilité d'inscription auto-confirmée (pour les environnements de test).
- **Monitoring** : Endpoints de métriques Prometheus et Health Checks pour Kubernetes.

2. Démarche et Méthodologie

Le développement du projet a suivi une approche agile et itérative, découpée en sprints techniques clairs pour assurer une progression constante et la qualité du code.

Phase 1 : Initialisation et Architecture

Mise en place de l'environnement de développement Next.js, configuration de Tailwind CSS et définition de la structure du projet. Création du schéma de base de données Supabase pour gérer les utilisateurs et les données de marché.

Phase 2 : Développement des Fonctionnalités Core

Implémentation des modules principaux : 1. **Dashboard** : Intégration de l'API CoinGecko pour récupérer les données en temps réel et affichage sous forme de tableaux et graphiques. 2. **Authentification** : Gestion complète des utilisateurs (inscription, connexion, protection des routes). 3. **Portefeuille** : Logique métier pour l'achat/vente de cryptomonnaies et suivi du solde.

Phase 3 : DevOps et Automatisation

Mise en place précoce des pipelines CI/CD pour automatiser les tests et le déploiement. Configuration de Docker pour la conteneurisation de l'application.

Phase 4 : Tests et Sécurité

Campagne de tests approfondie (unitaires, performances, sécurité) et audit de code via SonarCloud pour garantir la robustesse de l'application avant la livraison finale.

3. Pipelines CI/CD

L'automatisation du cycle de vie de l'application est gérée par GitHub Actions, divisée en trois workflows distincts.

Pipeline Principal (CI/CD)

Fichier : `.github/workflows/ci-cd.yml` Ce workflow se déclenche à chaque push sur la branche `main`.

1. **Tests et Qualité (Job: test)**
   - Installation des dépendances (Node.js 20).
   - Exécution des tests unitaires avec Jest et génération du rapport de couverture.
   - Analyse statique du code (Linting).
   - **SonarCloud Scan** : Envoi du rapport de couverture et analyse de la qualité du code (Bugs, Vulnerabilities, Code Smells).
2. **Construction (Job: build-and-push)**
   - Dépend du succès du job de test.
   - Connexion au registre de conteneurs GitHub (ghcr.io).
   - Extraction des métadonnées Docker.
   - Construction de l'image Docker et push vers le registre.
3. **Déploiement (Job: deploy)**
   - Dépend du succès du build.
   - S'exécute sur un **Self-Hosted Runner** configuré sur le cluster Kubernetes cible.
   - Mise à jour des déploiements Kubernetes (Deployment, Service, Ingress).
   - Vérification du statut du rollout.

Pipeline de Sécurité

Fichier : `.github/workflows/security.yml` Ce workflow scanne le projet de manière hebdomadaire ou à chaque modification critique.

- **Dependency Scan** : Audit npm et Snyk pour détecter les vulnérabilités dans les dépendances.

- **Container Scan** : Utilisation de Trivy pour scanner l'image Docker à la recherche de failles de sécurité (CVE).
- **OWASP ZAP** : Scan dynamique (DAST) de l'application web (configuré mais désactivé par défaut).

4. Tâches Planifiées (Cron Jobs)

Pour assurer la surveillance continue des marchés sans intervention utilisateur, un système de tâches planifiées a été mis en place.

Check Alerts Cron

Fichier : `.github/workflows/check-alerts.yml` * **Fréquence** : Toutes les 5 minutes (`*/5 * * * *`). * **Fonctionnement** : 1. Le workflow GitHub Actions déclenche une requête HTTP sécurisée vers l'endpoint API de l'application (`/api/cron/check-alerts`). 2. L'API récupère les derniers prix via CoinGecko. 3. Elle compare ces prix avec les alertes actives définies par les utilisateurs en base de données. 4. Si une condition est remplie (ex: BTC $>$ 50k), une notification est envoyée via Discord ou Email. * **Sécurité** : L'appel est protégé par un `CRON_SECRET` pour empêcher les exécutions non autorisées.

Backup Database Cron

Fichier : `k8s/backup-cronjob.yaml` Une ressource native Kubernetes `CronJob` exécute un backup complet de la base de données PostgreSQL chaque nuit à 2h00 du matin, conservant l'historique sur 7 jours.

5. Stratégie de Tests

La qualité de l'application est assurée par plusieurs niveaux de tests.

Tests Unitaires et d'Intégration

- **Outils** : Jest, React Testing Library.
- **Couverture** : Plus de 60 tests couvrant les composants critiques (Login, Charts, Notifications).
- **Exécution** : Automatisée dans le pipeline CI à chaque commit.

Tests de Performance (Load Testing)

- **Outil** : K6.
- **Scénario** : Simulation d'une montée en charge progressive jusqu'à 50 utilisateurs simultanés.
- **Métriques surveillées** : Temps de réponse (p95 $<$ 500ms) et taux d'erreur ($<$ 1%).
- **Script** : `tests/performance/load-test.js` teste les pages principales (Landing, Login, Dashboard).

Tests de Sécurité (SAST/DAST)

- **SonarCloud** : Analyse la maintenabilité et la sécurité du code source.
- **Snyk/Trivy** : Scannent les dépendances et l'image Docker.

6. Architecture Technique

Diagramme de Cas d'Utilisation

Diagramme de Classes (Base de Données)

Diagramme de Séquence : Collecte de Données & Alertes

Diagramme de Déploiement

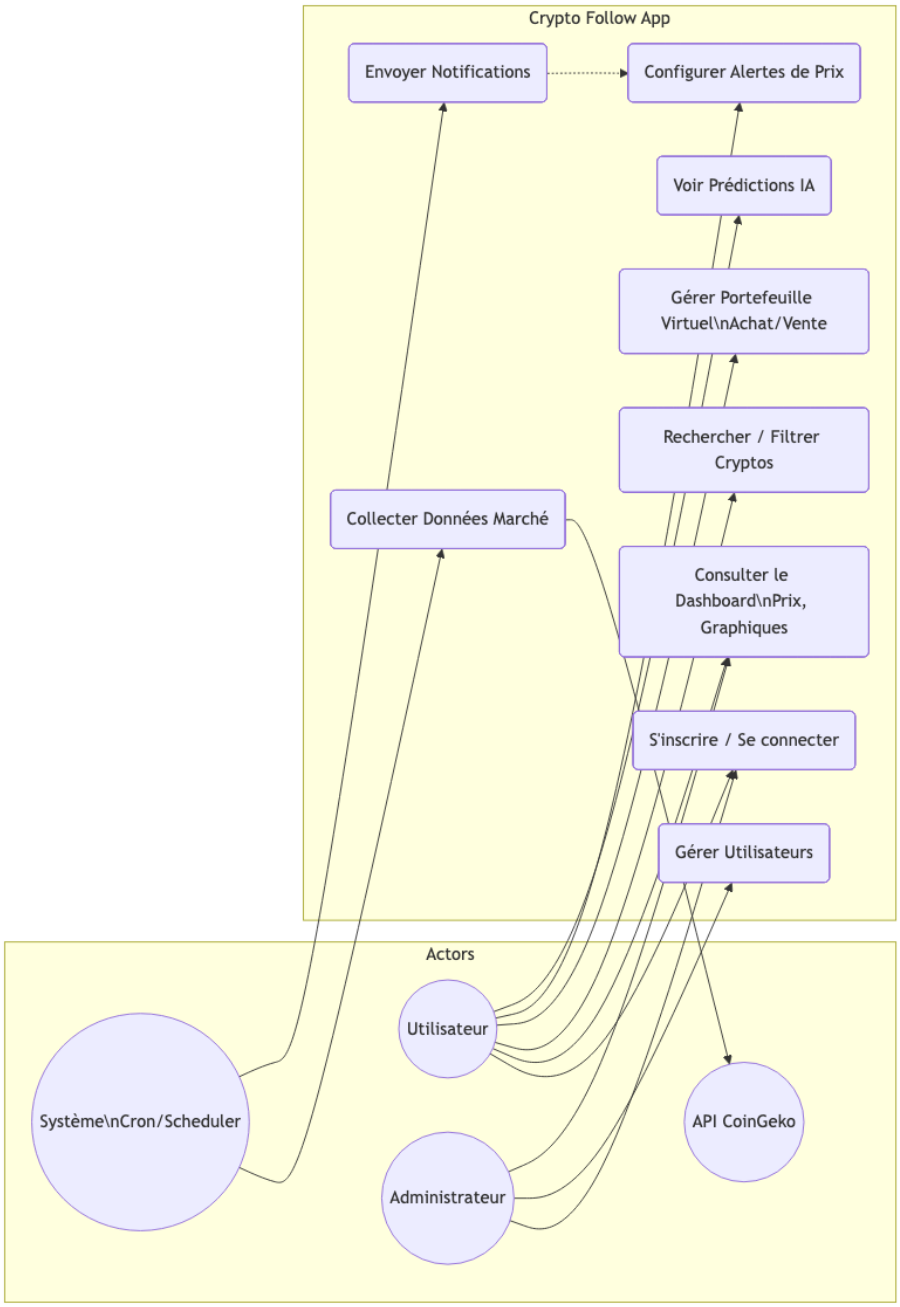

Figure 1: Use Case Diagram

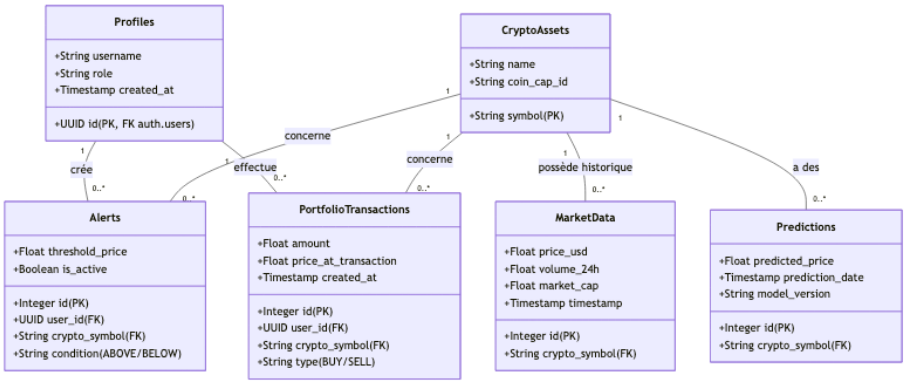

Figure 2: Class Diagram

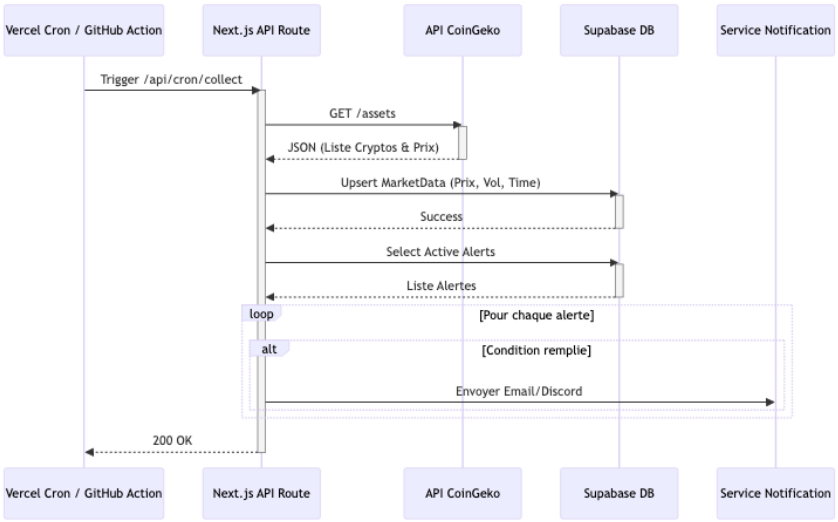

Figure 3: Sequence Diagram

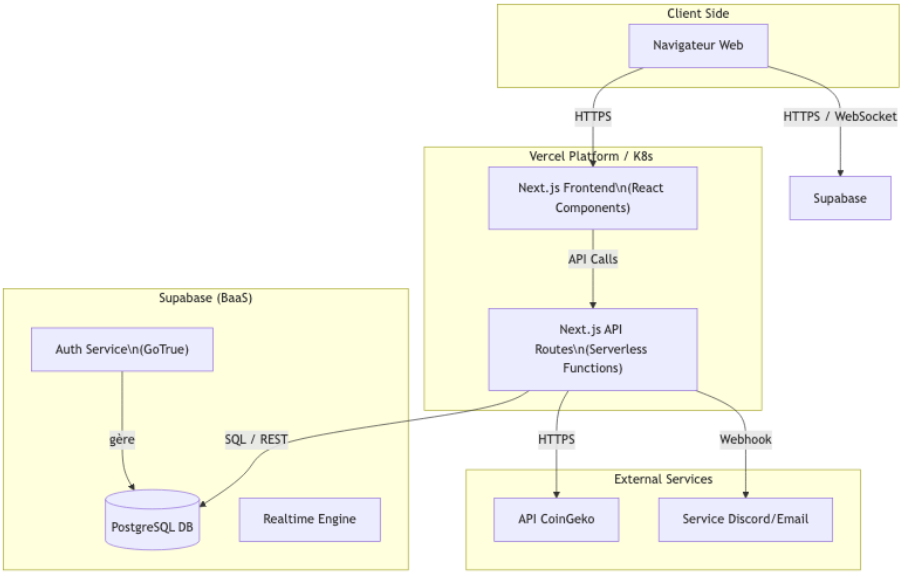

Figure 4: Deployment Diagram