CyPay
Framework d'Acteurs Distribués pour la Gestion de Transactions Crypto

Groupe 5
Rayan Sail
Adam Terrak
Mohamed Tounkara
Anthony Voisin
Lise Zheng

Programmation Web Avancée
Encadrés par Mme Zaouche Djaouida

December 26, 2025

Contexte et Objectifs

CyPay est une plateforme distribuée visant :

- Une gestion robuste des transactions de cryptomonnaies
- Une architecture microservices orientée acteurs
- Une intégration complète avec Spring Boot

L'objectif principal est la conception d'un **framework d'acteurs modulaire, scalable et sécurisé**.

Architecture Globale

La plateforme repose sur une architecture **microservices distribuée**.

- Services Spring Boot indépendants
- Communication via API REST

Au cœur de chaque service :

- Un **framework d'acteurs** assurant traitement asynchrone
- Parallélisme, résilience et découplage

Architecture Globale : Services

Cinq services principaux :

- **Transactions** : achat, vente, transfert
- **Wallet** : portefeuilles multi-devises
- **User** : authentification JWT
- **Logs** : traçabilité centralisée
- **Supervisor** : supervision et résilience

Cette séparation améliore maintenabilité et isolation des pannes.

Gestion des Transactions (Vue d'Ensemble)

Le service Transactions orchestre toute la logique métier.

- Endpoints REST (buy, sell, transfer)
- Traitement interne par agents spécialisés
- Exécution asynchrone et isolée

Chaque transaction est traçable et robuste.

Traitement d'une Transaction

Flux général :

- Requête HTTP $\rightarrow$ message interne typé
- Dispatch vers un agent (Buy, Sell, Transfer)
- Coordination avec Wallet et Logs

Les opérations critiques (débit/crédit) sont vérifiées et journalisées.

FRAMEWORK D'ACTEURS — Rôle Général

Le projet CyPay repose sur un **framework d'acteurs développé sur mesure**.

- Indépendant de la logique métier
- Réutilisable dans plusieurs microservices
- Responsable de la concurrence, de la résilience et de la scalabilité

Les services métier s'appuient exclusivement sur ce framework.

FRAMEWORK D'ACTEURS — Architecture Interne

Le framework est structuré autour de plusieurs briques :

- Gestion des acteurs et des messages
- Communication HTTP intégrée
- Sécurité, supervision et journalisation

Chaque brique est découplée et extensible.

FRAMEWORK D'ACTEURS — Abstraction Acteur

La classe **Acteur** représente un agent logiciel autonome :

- Possède un état interne
- Dispose d'une mailbox dédiée
- Traite les messages de manière séquentielle

Ce modèle simplifie la gestion de la concurrence.

FRAMEWORK D'ACTEURS — Cycle de Vie

Le framework prend en charge :

- La création dynamique des acteurs
- Leur initialisation contrôlée
- Leur arrêt et destruction propre

Les acteurs ne sont pas directement gérés par Spring.

FRAMEWORK D'ACTEURS — Mailbox et Exécution

Chaque acteur :

- Reçoit des messages dans une mailbox
- Les traite de manière séquentielle
- Ne partage pas son état avec d'autres acteurs

La cohérence et le thread-safety sont garantis.

FRAMEWORK D'ACTEURS — Blocage et Déblocage

Le framework permet :

- Le blocage temporaire d'un acteur
- L'accumulation contrôlée des messages
- La reprise du traitement sans perte d'état

Aucun verrou explicite n'est nécessaire.

FRAMEWORK D'ACTEURS — Messages

La communication repose sur des messages typés :

- Classe `Message`
- Découplage émetteur / récepteur
- Envoi non bloquant

Il s'agit du mode de communication principal du framework.

FRAMEWORK D'ACTEURS — Communication HTTP

Le framework intègre la communication synchrone REST :

- **HttpReceiver** pour les requêtes entrantes
- **ActeurHttpClient** pour les appels sortants

Les requêtes HTTP sont transformées en messages internes.

FRAMEWORK D'ACTEURS — Communication Inter-Microservices

Le framework permet :

- Des échanges synchrones entre microservices
- Une orchestration asynchrone côté acteur

Les threads web ne sont jamais bloqués par le traitement métier.

FRAMEWORK D'ACTEURS — Supervision

Un superviseur central :

- Surveille l'état des acteurs
- Détecte les erreurs et blocages
- Reçoit les notifications d'anomalies

Chaque acteur est supervisé de manière isolée.

FRAMEWORK D'ACTEURS — Tolérance aux Pannes

En cas d'erreur :

- L'acteur fautif est isolé
- Un redémarrage automatique est possible
- Le reste du système continue à fonctionner

Les pannes globales sont évitées.

FRAMEWORK D'ACTEURS — Scalabilité Dynamique

Les acteurs critiques sont gérés par des pools dynamiques :

- Ajustement automatique du nombre d'instances
- Basé sur la taille des files de messages

La scalabilité est gérée au niveau applicatif.

FRAMEWORK D'ACTEURS — Paramétrage des Pools

Chaque pool est défini par :

- `minActors`
- `maxActors`
- `highWatermark`
- `lowWatermark`

Ce mécanisme équilibre performance et consommation de ressources.

FRAMEWORK D'ACTEURS — Logs et Traçabilité

Le framework intègre un système de journalisation centralisé :

- Acteur dédié : **ActeurLogger**
- Écriture console et base de données
- Format standardisé et exploitable

Toutes les actions utilisateur sont tracées.

FRAMEWORK D'ACTEURS — Conformité à l'Énoncé

Le framework répond explicitement à :

- Gestion des acteurs
- Communication synchrone et asynchrone
- Supervision et tolérance aux pannes
- Scalabilité dynamique
- Traçabilité des actions utilisateur

Le framework constitue la valeur ajoutée principale du projet.

COMPARAISON AVEC AKKA — Points forts de CyPay

CyPay adapte le modèle d'Akka aux microservices Spring Boot :

- **Intégration Spring Boot** : Annotations, injection de dépendances, sécurité JWT.
- **Modèle hybride** : REST pour les APIs, acteurs asynchrones en interne.
- **Scalabilité automatique** : Pools dynamiques et mécanisme de Watermarks.

COMPARAISON AVEC AKKA — Limites et différences

Choix assumés pour plus de simplicité :

- **Pas de transparence de localisation** : Local et distant explicitement distingués.
- **Persistance limitée** : Pas de restauration automatique de l'état.
- **Supervision centralisée** : Pas de hiérarchie parent-enfant stricte.

Conclusion

CyPay privilégie une **solution légère, lisible et orientée Web**, au détriment de certaines fonctionnalités avancées d'Akka.