Coinflux - Microservices-Based Fintech Wallet

Overview

Coinflux is a production-grade Fintech Wallet Application built using a microservices architecture. The system allows users to register, login, manage wallets, perform fund transfers, raise disputes, and receive real-time notifications. It uses JWT-based authentication, API Gateway pattern, and internal service communication via RabbitMQ.

Live Swagger Documentation

Coinflux API Docs

Architecture Diagram

Microservices

1. User Service

- Handles user registration and authentication.
- Generates JWT tokens on successful login.

Tech Stack: Node.js, Express, MySQL, Sequelize

2. Wallet Service

- Manages wallets: balance inquiry, add funds, withdraw, and transfer.
- Listens to internal RabbitMQ messages to notify about actions.

Tech Stack: Node.js, Express, MySQL, Sequelize

3. Dispute Service

- Allows users to raise disputes for transactions.
- Supports auto-review jobs to update dispute statuses.

Tech Stack: Node.js, Express, MySQL, Sequelize, Node-Cron

4. Notification Service

- Listens to RabbitMQ messages to send transactional emails via Mailtrap.

Tech Stack: Node.js, Express, RabbitMQ, Mailtrap

5. API Gateway

- Central entry point for all services.
- Proxies requests to respective services.
- Contains Swagger documentation for all APIs.

Tech Stack: Node.js, Express, http-proxy-middleware, Swagger

Key Features

- ✅ JWT-based authentication
- 📶 Role-based protected routes
- 🔁 RabbitMQ message queues
- ◀️ Email notification system
- 📘 Swagger UI for complete API documentation
- 🐳 Docker-based containerization
- 🚀 Deployed to Render with service-level separation

Folder Structure

```
coinflux/
├── api-gateway
├── user-service
├── wallet-service
├── dispute-service
├── notification-service
├── docker-compose.yml
└── swagger
```

Installation (Local)

```
git clone <repo-url>
cd coinflux
docker-compose up --build
```

Visit Swagger Docs: `http://localhost:8080/api-docs`

Deployment (Render)

Each service is deployed independently to Render. The API Gateway aggregates all endpoints and serves Swagger UI.

Technologies Used

- Node.js
- Express.js
- MySQL + Sequelize ORM
- RabbitMQ
- Swagger
- Docker + Docker Compose
- Mailtrap
- Render

How This Showcases My Skills

- ✅Hands-on experience in **Microservices Architecture**
- ✅Practical implementation of **API Gateway Pattern**
- ✅Integration of **JWT Authentication**, **Proxying**, and **Internal Communication (RabbitMQ)**
- ✅Experience deploying services with **Docker** and **Render**
- ✅Good understanding of **API Documentation (Swagger)**
- ✅Demonstrates ability to build **Scalable and Modular Applications**

Future Improvements

- Add Redis caching for frequently accessed endpoints.
- Introduce MongoDB for notifications logs.
- Rate limiting via API Gateway.
- Helm charts for Kubernetes deployment.

License