Cryptocurrency Exchange Backend

This project provides a basic backend for a cryptocurrency exchange, featuring order placement, order book management, real-time market data updates via WebSockets, and a simulated trade settlement process using RabbitMQ and a mock blockchain interaction.

Project Structure

- api.py: The FastAPI application that handles API endpoints for orders and order book data, and a WebSocket for real-time market data.
- consumer.py: A RabbitMQ consumer that simulates the settlement of trades by processing messages from a queue and interacting with a mock blockchain (Ethereum testnet).
- Dockerfile: Defines the Docker image for containerizing the FastAPI application.
- requirements.txt: Lists all Python dependencies required for the project.
- .env: (Not provided, but assumed) This file would contain environment variables such as ETH_TESTNET_URL, SETTLEMENT_CONTRACT_ADDRESS, SETTLEMENT_CONTRACT_ABI, and RABBITMQ_URI.

Features

- **Order Placement**: Users can place limit and market orders for various trading pairs.
- **Order Book**: Provides a real-time view of bids and asks for specified trading pairs.
- **Real-time Market Data**: WebSocket endpoint for clients to receive live order book updates.
- **Trade Settlement (Simulated)**: A separate consumer process handles trade settlement by consuming messages from a RabbitMQ queue and interacting with a mock blockchain.
- **Containerization**: Dockerfile included for easy deployment and environment consistency.

Setup and Installation

Prerequisites

- **Docker**: For containerized deployment.
- **Python 3.11**: If running locally without Docker.
- **RabbitMQ**: Message broker for trade settlement.
- **Ethereum Testnet Node/Provider**: For blockchain interaction (e.g., Infura, Alchemy).

Environment Variables

Create a .env file in the root directory of the project and populate it with the following:

ETH_TESTNET_URL="YOUR_ETHEREUM_TESTNET_RPC_URL"
SETTLEMENT_CONTRACT_ADDRESS="YOUR_SETTLEMENT_SMART_CONTRACT_ADDRESS"

SETTLEMENT_CONTRACT_ABI='[{"your":"contract"},{"abi":"here"}]'
RABBITMQ_URI="amqp://guest:guest@localhost:5672/"

- ETH_TESTNET_URL: The URL of your Ethereum testnet RPC provider (e.g., https://sepolia.infura.io/v3/YOUR_PROJECT_ID).
- SETTLEMENT_CONTRACT_ADDRESS: The address of your deployed settlement smart contract on the Ethereum testnet.
- SETTLEMENT_CONTRACT_ABI: The JSON ABI of your settlement smart contract, as a string.
- RABBITMQ_URI: The connection URI for your RabbitMQ instance.

Running with Docker (Recommended)

1. **Build the Docker image**:
   docker build -t crypto-exchange-backend .

2. **Run the Docker container**:
   docker run -p 8000:8000 --env-file ./.env crypto-exchange-backend

   This will start the FastAPI application.

Running Locally (Without Docker)

1. **Install dependencies**:
   pip install -r requirements.txt

2. **Run the FastAPI application**:
   uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

   (Note: app.main:app assumes your main FastAPI instance is named app within a main.py file inside an app directory. Adjust if your project structure differs, e.g., api:app if api.py is at the root.)
3. Run the RabbitMQ consumer:
   Open a separate terminal and run:
   python consumer.py

API Endpoints

1. Place an Order

- **URL**: /api/v1/order
- **Method**: POST
- **Body (JSON)**:
  {

```
  "order_id": "unique-order-id-123",
  "user_id": "user-abc",
  "pair": "BTC-USD",
  "order_type": "limit", // or "market"
  "side": "buy",        // or "sell"
  "price": 30000.00,    // Required for limit orders, ignored for market orders
  "amount": 0.001
}
```

- **Response**:
```
{
  "message": "Order submitted successfully",
  "order_id": "unique-order-id-123"
}
```

2. Get Order Book

- **URL**: /api/v1/orderbook/{pair}
- **Method**: GET
- **Path Parameter**:
  - pair: The trading pair (e.g., BTC-USD).
- **Response**:
```
{
  "bids": [
    {"price": 29999.50, "amount": 0.005},
    {"price": 29999.00, "amount": 0.01}
  ],
  "asks": [
    {"price": 30000.50, "amount": 0.003},
    {"price": 30001.00, "amount": 0.008}
  ]
}
```

3. Real-time Market Data (WebSocket)

- **URL**: /ws/marketdata
- **Protocol**: WebSocket
- **Description**: Connect to this endpoint to receive continuous JSON updates of the order book for BTC-USD.
- **Message Format (received from server)**:
```
{
  "type": "orderbook_update",
  "data": {
```

```
        "bids": [...],
        "asks": [...]
    }
}
```

Development Notes

- **Authentication & Balance Checks**: The place_order endpoint currently has TODO comments for robust user authentication and balance checks. These are critical for a production system.
- **Binance Integration**: The api.py includes commented-out code for integrating with Binance's live data stream. This can be uncommented and implemented to provide real-time pricing from an external source.
- **Blockchain Interaction**: The consumer.py currently simulates on-chain settlement. In a real scenario, you would integrate actual web3.py calls to sign and send transactions to your smart contract.
- **Error Handling**: Basic error handling is in place, but could be further enhanced for production robustness.
- **Database**: This project uses an in-memory order_books dictionary for simplicity. A production-grade exchange would require a persistent database (e.g., MongoDB, PostgreSQL) for orders, user balances, and trade history.

Contributing

Contributions are welcome! Please feel free to open issues or submit pull requests.