Blockchain Smart Contracts and Deployment

This section outlines the smart contracts for a hybrid trading system and the scripts used to deploy and interact with them on an Ethereum-compatible blockchain. These contracts form the on-chain settlement layer, working in conjunction with an off-chain matching engine.

Project Structure

- Settlement.sol: The Solidity smart contract responsible for settling trades by transferring mock ERC20 tokens between buyers and sellers. It includes an Ownable pattern for access control and a mechanism to prevent double-settlements.
- deploy.js: A Hardhat script that deploys the MockERC20 token contracts (USDT and WBTC) and the Settlement contract to a specified network. It also mints initial tokens to the deployer's address for testing purposes and outputs the deployed contract addresses.
- hardhat.config.js: The Hardhat configuration file. It specifies the Solidity compiler version, defines network configurations (e.g., Sepolia testnet), and integrates with dotenv to load environment variables for sensitive information like RPC URLs and private keys.
- package.json: Manages the project's dependencies, scripts, and metadata for the Hardhat development environment. It lists required packages like @nomicfoundation/hardhat-toolbox and @openzeppelin/contracts.

Smart Contracts

MockERC20.sol

A simplified ERC20 token contract used for simulation and testing. It inherits from OpenZeppelin's ERC20 and Ownable contracts, providing basic token functionalities (transfer, balance) and owner-restricted minting. It's designed to represent tradable assets like USDT and WBTC in a test environment.

Settlement.sol

This contract serves as the core on-chain settlement layer. Its primary function, settleTrade, facilitates the transfer of tokenA from the buyer to the seller and tokenB from the seller to the buyer. Key features include:

- **onlyOwner Modifier**: Ensures that only the authorized backend service (the contract owner) can initiate trade settlements.
- **Double-Settlement Prevention**: Uses a mapping (settledTrades) to track processed tradeIds, preventing a single trade from being settled multiple times.
- **Event Emission**: Emits a TradeSettled event upon successful settlement, providing an

on-chain log for off-chain applications to monitor.

Deployment and Configuration

Prerequisites

- **Node.js & npm**: For running Hardhat scripts.
- **Hardhat**: Ethereum development environment.
- **OpenZeppelin Contracts**: Standard library for secure smart contract development.
- **.env file**: Contains sensitive information.

Environment Variables

Before deployment, create a .env file in the root directory (one level up from the blockchain directory, as configured in hardhat.config.js) with the following:

```
ETH_TESTNET_URL="YOUR_ETHEREUM_TESTNET_RPC_URL"
PRIVATE_KEY="YOUR_METAMASK_PRIVATE_KEY"
ETHERSCAN_API_KEY="YOUR_ETHERSCAN_API_KEY"
```

- ETH_TESTNET_URL: Your Ethereum testnet RPC URL (e.g., from Infura or Alchemy).
- PRIVATE_KEY: The private key of the Ethereum account you'll use for deploying contracts. **Do not use a private key for an account with real funds in a public repository.**
- ETHERSCAN_API_KEY: Your API key for Etherscan (or a similar block explorer) for contract verification.

Installation

1. **Navigate to the blockchain directory**:
   ```
   cd blockchain
   ```
2. **Install Node.js dependencies**:
   ```
   npm install
   ```

Deployment

To deploy the contracts to your configured network (e.g., Sepolia), run the deployment script:

```
npx hardhat run scripts/deploy.js --network sepolia
```

Upon successful deployment, the deploy.js script will output the deployed addresses for Settlement, MockUSDT, and MockWBTC. These addresses are crucial and should be copied into the SETTLEMENT_CONTRACT_ADDRESS, MOCK_USDT_ADDRESS, and MOCK_WBTC_ADDRESS variables in the .env file of your backend application (api.py, consumer.py).

Usage

After deployment, the Settlement contract's settleTrade function can be called by the initialOwner (typically your backend's settlement service) to finalize off-chain trades on the blockchain. The MockERC20 tokens are used to simulate asset transfers.

Development Notes

- **Security**: The Settlement.sol contract uses the onlyOwner modifier for critical functions, which centralizes control. In a highly decentralized system, alternative access control mechanisms (e.g., multi-sig wallets, DAOs) might be considered.
- **Token Approvals**: For the Settlement contract to transfer tokens, both the buyer and seller must have *approved* the Settlement contract to spend their respective tokenA and tokenB amounts prior to calling settleTrade. This approval mechanism is standard for ERC20 transferFrom operations.
- **Gas Costs**: Real-world deployments involve transaction fees (gas). The current scripts do not explicitly handle gas estimation or optimization, which would be crucial for production.
- **Testing**: While the deploy.js script mints tokens for testing, comprehensive unit and integration tests for the smart contracts are essential. Hardhat provides robust testing frameworks.