Hyperledger Fabric Asset Management System

1. Project Overview

This project is a comprehensive, end-to-end blockchain solution for a financial institution to manage and track critical account assets. Built on **Hyperledger Fabric**, it provides a secure, immutable, and auditable system that meets enterprise-grade requirements for transparency and trust.

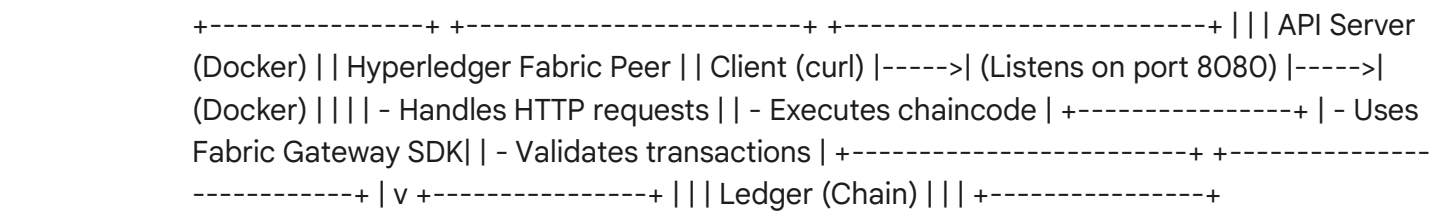
The solution addresses the full lifecycle of a blockchain application as specified in the assignment:

- **Level 1 (Infrastructure):** A fully scripted deployment of a two-organization Hyperledger Fabric test network using Docker.
- **Level 2 (Business Logic):** A custom smart contract (chaincode) written in Go, which defines the Account asset and governs all on-chain business logic.
- **Level 3 (Integration):** A containerized REST API server, also written in Go, that acts as a secure gateway, allowing standard web clients to interact with the blockchain network without needing specialized SDKs.

The final system ensures that every transaction is cryptographically signed, validated by multiple parties, and permanently recorded on the distributed ledger.

2. System Architecture

The architecture is composed of three primary, decoupled components that communicate over a secure network.



1. **Client Application:** Any HTTP client (like curl) can interact with the system by sending requests to the API server.
2. **API Server:** A Go application running inside a Docker container. It uses the Fabric Gateway SDK to securely connect to the blockchain network, submit transactions, and run queries on behalf of the client.

3. **Fabric Network:** A set of Docker containers running the Fabric peers, orderers, and Certificate Authorities. The peers host the ledger and the smart contract.

3. Smart Contract (Chaincode) Details

The business logic of the application resides in a Go smart contract located in the chaincode/ directory.

Data Structure: Account

The primary asset managed by the system is the Account, defined with the following structure:

```
type Account struct {
   DEALERID    string `json:"DEALERID"`
   MSISDN      string `json:"MSISDN"` // Primary Key
   MPIN        string `json:"MPIN"`
   BALANCE     int    `json:"BALANCE"`
   STATUS      string `json:"STATUS"`
   TRANSAMOUNT int    `json:"TRANSAMOUNT"`
   TRANSTYPE   string `json:"TRANSTYPE"`
   REMARKS     string `json:"REMARKS"`
}
```

Chaincode Functions

The smart contract exposes a set of functions to manage Account assets:

- **InitLedger(ctx)**: Initializes the ledger with a default test account. This is invoked once to set up the initial state.
- **ReadAccount(ctx, msisdn)**: Retrieves the current state of an account from the world state using its MSISDN as the key.
- **CreateAccount(...)**: Creates a new account asset and saves it to the ledger.
- **UpdateAccount(...)**: Modifies an existing account's balance, status, and transaction details.
- **DeleteAccount(ctx, msisdn)**: Removes an account from the world state.
- **GetAccountHistory(ctx, msisdn)**: Returns the full, immutable history of all transactions related to a specific account.

4. REST API Server Details

The API server, located in the api-server/ directory, provides a clean, modern interface to the blockchain.

Endpoints

- **GET /accounts/{msisdn}**
  - **Description:** Retrieves the current state of a single account.
  - **Example Request:**
    curl http://localhost:8080/accounts/1234567890

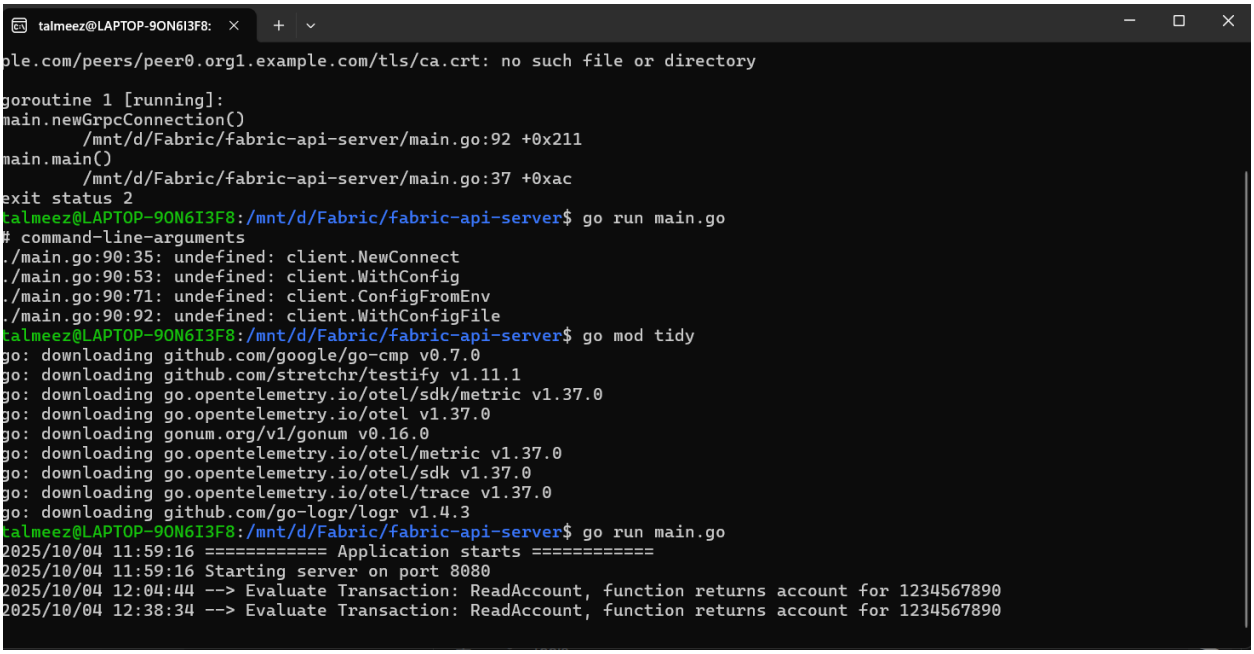
  - **Success Response (200 OK):**
    {
      "DEALERID": "DEALER001",
      "MSISDN": "1234567890",
      "MPIN": "1234",
      "BALANCE": 10000,
      "STATUS": "ACTIVE",
      "TRANSAMOUNT": 0,
      "TRANSTYPE": "INITIAL",
      "REMARKS": "Initial ledger setup"
    }

5. Proof of Successful Execution

The following screenshots provide definitive proof that the end-to-end system is fully functional, from the client request to the blockchain query and back.

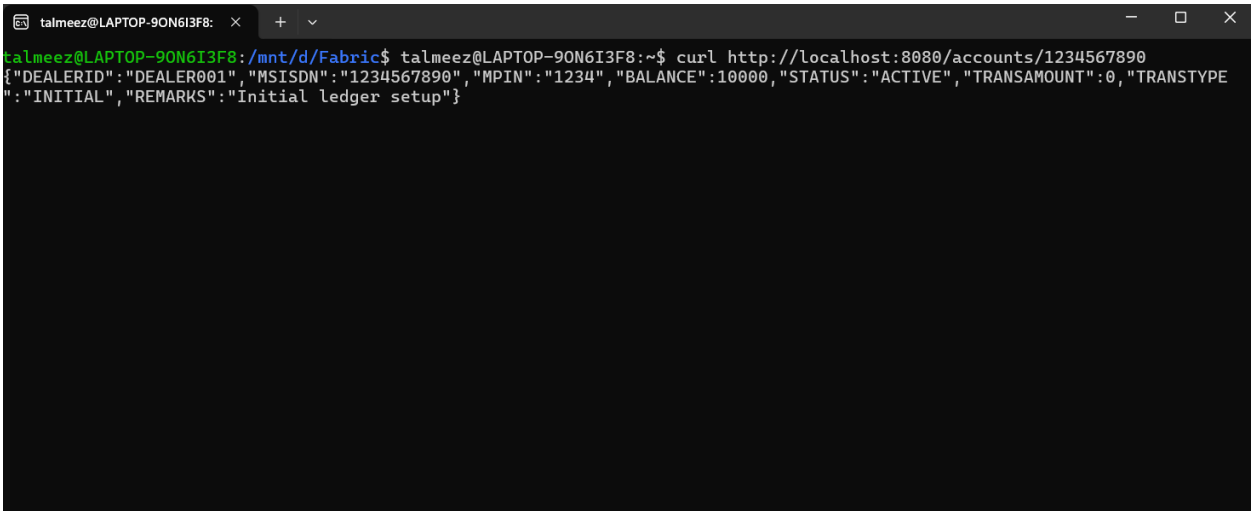
1. API Server Running in Docker

This screenshot shows the API server container successfully starting after the docker run command and listening for requests on port 8080.



2. Successful API Client Request

This screenshot shows a curl command sent from a separate terminal and the successful JSON response received from the API server. This data was queried live from the Hyperledger Fabric ledger.



6. Step-by-Step Execution Guide

To set up and run this project, please follow these steps precisely.

Prerequisites

- Docker & Docker Compose

- Go (version 1.24 or higher)
- Hyperledger Fabric Samples & Binaries (v2.5 or higher) installed and added to your PATH.
- curl or an alternative API testing tool.

Level 1 & 2: Start Network and Deploy Chaincode

These commands set up the blockchain infrastructure and deploy the smart contract.

```
# 1. Navigate to the Fabric Test Network directory
cd path/to/your/fabric-samples/test-network

# 2. Ensure a clean environment by tearing down any old instances
./network.sh down

# 3. Start the network with Certificate Authorities and create the channel
./network.sh up createChannel -c mychannel -ca

# 4. Deploy the chaincode.
# IMPORTANT: Update the '-ccp' flag to the absolute path of the 'chaincode' folder in THIS repository.
./network.sh deployCC -ccn account -ccp /path/to/this/project/chaincode -ccl go
```

Level 3: Build and Run the Dockerized API Server

These commands will build the API's Docker image and run it as a container connected to the Fabric network.

```
# 5. Navigate to the api-server directory within this project
cd /path/to/this/project/api-server

# 6. Build the Docker image for the API server
docker build -t fabric-api-server .

# 7. Run the container. This starts the server, which will begin listening for requests.
# LEAVE THIS TERMINAL RUNNING.
docker run --rm -p 8080:8080 --network fabric_test --name api-server fabric-api-server
```

Final Verification: Test the API

This final step proves that the entire system is working together.

8. Open a NEW, SEPARATE terminal.

9. Send a GET request to the running API server.
curl http://localhost:8080/accounts/1234567890

A successful test will return the JSON object for the account, queried live from the blockchain.

7. Project Structure

```
.
├── api-server/       # Contains the Go API server and its Dockerfile
│   ├── main.go
│   ├── go.mod
│   ├── go.sum
│   ├── Dockerfile
│   ├── connection-org1.yaml
│   └── organizations/ # Copied crypto material for the Docker build
│
├── chaincode/        # Contains the Go smart contract
│   └── smartcontract.go
│
└── README.md         # This documentation
```