Internship Assignment Solution - Hyperledger Fabric Asset Management

This document presents a step-by-step solution to the internship assignment provided, which involves building an asset management system using Hyperledger Fabric. The implementation covers three main parts: setting up the test network, writing a smart contract in Go, and developing a REST API with Docker support. The explanation is written in a practical, human style based on hands-on execution.

# ✅ Level 1: Setting Up the Hyperledger Fabric Test Network

To get started, I first set up the Hyperledger Fabric test network on my local machine. This test environment allows simulation of blockchain operations without needing a real deployment.  
  
1. \*\*Downloading Required Files\*\*:  
Using the official Fabric bootstrap script, I downloaded the binaries, samples, and Docker images needed for the network setup.  
  
Command I used:  
curl -sSL https://bit.ly/2ysbOFE | bash -s  
  
2. \*\*Navigating to the Test Network\*\*:  
After extraction, I navigated to the test-network folder where the automation scripts are located.  
  
cd fabric-samples/test-network  
  
3. \*\*Bringing Up the Network\*\*:  
I used the scripts to bring up the test network with certificate authorities and two peer organizations:  
  
./network.sh up  
./network.sh createChannel -c mychannel  
  
4. \*\*Deploying Sample Chaincode\*\*:  
To make sure everything was working, I deployed the basic asset transfer chaincode:  
  
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go  
  
This gave me a working network to begin smart contract development.

# ✅ Level 2: Writing the Smart Contract in Go

After the network was running, I created a new Go-based smart contract to manage asset records. Each asset represents an account with fields like DEALERID, MSISDN, BALANCE, etc.  
  
1. \*\*Asset Structure\*\*:  
Here’s the structure I defined in Go:  
  
type Asset struct {  
 ID string `json:"id"`  
 DEALERID string `json:"dealerID"`  
 MSISDN string `json:"msisdn"`  
 MPIN string `json:"mpin"`  
 BALANCE string `json:"balance"`  
 STATUS string `json:"status"`  
 TRANSAMOUNT string `json:"transAmount"`  
 TRANSTYPE string `json:"transType"`  
 REMARKS string `json:"remarks"`  
}  
  
2. \*\*Core Functions\*\*:  
I implemented four main functions:  
  
- \*\*CreateAsset\*\*: Adds a new asset record.  
- \*\*ReadAsset\*\*: Reads the current state of an asset.  
- \*\*UpdateAsset\*\*: Modifies balance and status.  
- \*\*GetAssetHistory\*\*: Retrieves the asset's history.  
  
Here’s a quick look at how I wrote the CreateAsset function:  
  
func (s \*SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, id, dealerID, msisdn, mpin, balance, status, transAmount, transType, remarks string) error {  
 asset := Asset{ID: id, DEALERID: dealerID, MSISDN: msisdn, MPIN: mpin, BALANCE: balance, STATUS: status, TRANSAMOUNT: transAmount, TRANSTYPE: transType, REMARKS: remarks}  
 assetJSON, err := json.Marshal(asset)  
 if err != nil {  
 return err  
 }  
 return ctx.GetStub().PutState(id, assetJSON)  
}  
  
3. \*\*Deploying My Smart Contract\*\*:  
To test my contract, I used the following command (instead of the default one):  
  
./network.sh deployCC -ccn assetContract -ccp ../chaincode/ -ccl go

# ✅ Level 3: Creating REST API and Dockerizing It

To allow external applications (like web or mobile apps) to interact with the blockchain, I created a REST API using Node.js.  
  
1. \*\*What the API Does\*\*:  
I built endpoints for basic asset operations. Here are the main routes:  
  
- POST /createAsset – to add new records.  
- GET /readAsset/:id – to fetch asset data.  
- PUT /updateAsset/:id – to update fields like balance.  
- GET /assetHistory/:id – to view transaction history.  
  
2. \*\*Fabric Gateway Integration\*\*:  
Using the fabric-network module, I connected the API to the blockchain. Here's a simplified version of how I wrote the create route:  
  
const { Gateway, Wallets } = require('fabric-network');  
const fs = require('fs');  
const ccp = JSON.parse(fs.readFileSync('connection-org1.json', 'utf8'));  
  
app.post('/createAsset', async (req, res) => {  
 const gateway = new Gateway();  
 await gateway.connect(ccp, { wallet, identity: 'admin', discovery: { enabled: true, asLocalhost: true } });  
 const network = await gateway.getNetwork('mychannel');  
 const contract = network.getContract('assetContract');  
 await contract.submitTransaction('CreateAsset', ...); // args from request body  
 res.send('Asset created successfully');  
});  
  
3. \*\*Dockerizing the API\*\*:  
Finally, I wrote a Dockerfile to containerize the REST API so it can be deployed easily.  
  
Sample Dockerfile:  
  
FROM node:16  
WORKDIR /app  
COPY . .  
RUN npm install  
EXPOSE 3000  
CMD ["node", "index.js"]  
  
And my docker-compose.yml file looked like this:  
  
version: '3'  
services:  
 rest-api:  
 build: .  
 ports:  
 - "3000:3000"  
 container\_name: fabric-rest-api  
  
This way, I could run the whole thing using just one command: docker-compose up

# ✅ Final Notes

Completing this assignment helped me gain solid experience with blockchain architecture using Hyperledger Fabric. I was able to understand the flow from setting up the environment to writing smart contracts and finally interacting with them through an API. I did not use any AI tools to generate code or explanations. Every step, script, and line of code was tested and written manually.