CHAINCODE (AGRIFOOD.GO)

This appendix contains the complete Go code for the `agrifood` smart contract. The code includes functions for creating, updating, and querying products, with built-in access control and robust error handling to ensure data integrity and security on the ledger.

```go
package main

import (
    "encoding/json"
    "fmt"

    "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

type SmartContract struct {
    contractapi.Contract
}

type Product struct {
    ID       string `json:"id"`
    Owner    string `json:"owner"`  // Org1MSP, Org2MSP, Org3MSP
    Status   string `json:"status"` // CREATED, SHIPPED, SOLD
    Location string `json:"location"`
    Quality  string `json:"quality"`
}

// CreateProduct - Only Org1 (Farmer) can create
func (s *SmartContract) CreateProduct(ctx contractapi.TransactionContextInterface, id, location, quality string) error {
    mspID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil || mspID != "Org1MSP" {
        return fmt.Errorf("only Org1 (Farmer) can create products")
    }

    exists, _ := s.ProductExists(ctx, id)
    if exists {
        return fmt.Errorf("product %s already exists", id)
    }

    product := Product{
        ID:       id,
        Owner:    mspID,
        Status:   "CREATED",
        Location: location,
        Quality:  quality,
    }

    productJSON, _ := json.Marshal(product)
    return ctx.GetStub().PutState(id, productJSON)
}

// UpdateProductStatus - Only Org2 (Retailer) can update, and only if current owner is Org1MSP
func (s *SmartContract) UpdateProductStatus(ctx contractapi.TransactionContextInterface, id, status, location string) error {
    mspID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil || mspID != "Org2MSP" {
        return fmt.Errorf("only Org2 (Retailer) can update product status")
    }

    productJSON, err := ctx.GetStub().GetState(id)
    if err != nil || productJSON == nil {
        return fmt.Errorf("product %s does not exist", id)
    }

    var product Product
    _ = json.Unmarshal(productJSON, &product)

    if product.Owner != "Org1MSP" {
        return fmt.Errorf("product %s is not owned by Org1; cannot be updated by Org2", id)
    }

    product.Status = status
    product.Location = location
    product.Owner = mspID // Transfer ownership to Org2

    updatedJSON, _ := json.Marshal(product)
    return ctx.GetStub().PutState(id, updatedJSON)
}

// PurchaseProduct - Only Org3 (Customer) can purchase, and only if owned by Org2MSP
func (s *SmartContract) PurchaseProduct(ctx contractapi.TransactionContextInterface, id string) error {
    mspID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil || mspID != "Org3MSP" {
        return fmt.Errorf("only Org3 (Customer) can purchase products")
    }

    productJSON, err := ctx.GetStub().GetState(id)
    if err != nil || productJSON == nil {
        return fmt.Errorf("product %s does not exist", id)
    }

    var product Product
    _ = json.Unmarshal(productJSON, &product)

    if product.Owner != "Org2MSP" {
        return fmt.Errorf("product %s is not owned by Org2; cannot be sold to Org3", id)
    }

    product.Status = "SOLD"
    product.Owner = mspID

    updatedJSON, _ := json.Marshal(product)
    return ctx.GetStub().PutState(id, updatedJSON)
}

// ProductExists utility function
func (s *SmartContract) ProductExists(ctx contractapi.TransactionContextInterface, id string) (bool, error) {
    data, err := ctx.GetStub().GetState(id)
    return data != nil, err
}

// GetProduct - Anyone can read product by ID
func (s *SmartContract) GetProduct(ctx contractapi.TransactionContextInterface, id string) (*Product, error) {
    productJSON, err := ctx.GetStub().GetState(id)
    if err != nil || productJSON == nil {
        return nil, fmt.Errorf("product %s not found", id)
    }

    var product Product
    _ = json.Unmarshal(productJSON, &product)
    return &product, nil
}

// GetAllProducts - Return all products (open query for simplicity)
func (s *SmartContract) GetAllProducts(ctx contractapi.TransactionContextInterface) ([]*Product, error) {
    iterator, err := ctx.GetStub().GetStateByRange("", "")
    if err != nil {
        return nil, err
    }
    defer iterator.Close()
```

```go

    var products []*Product
    for iterator.HasNext() {
        item, err := iterator.Next()
        if err != nil {
                    return nil, err
        }
        var product Product
        _ = json.Unmarshal(item.Value, &product)
        products = append(products, &product)
    }

    return products, nil
}

func main() {
    chaincode, err := contractapi.NewChaincode(new(SmartContract))
    if err != nil {
        panic("Error creating chaincode: " + err.Error())
    }
    if err := chaincode.Start(); err != nil {
        panic("Error starting chaincode: " + err.Error())
    }
}
```

NETWORK SETUP AND CLI COMMANDS

This appendix provides a complete, step-by-step record of the commands used to deploy and test the Agri-food Supply Chain Management system. All commands are run from the fabric-samples/test-network directory unless specified otherwise.

Phase 1: Environment and Network Initialization

These commands prepare the environment and bring up the core three-organization network.

1. **Clean up previous runs (if any):**

   ```
   ./network.sh down
   ```

2. **Launch a two-organization network with a channel:**

   ```
   ./network.sh up createChannel -ca
   ```

3. **Add the third organization (Org3) to the network:**

   Bash

   ```
   ./addOrg3.sh up -c mychannel
   ```

   (Note: This command is run from the addOrg3 directory.)

4. **Set the environment variables for Fabric binaries:**

   ```
   export PATH=$PATH:~/agrifood/fabric-samples/bin
   ```

5. **Set the configuration path:**

   ```
   export FABRIC_CFG_PATH=$PWD/../config
   ```

Phase 2: Chaincode Installation and Deployment

This phase covers the entire lifecycle of the agrifood chaincode, from packaging to committing it on the channel.

6. **Package the chaincode:**

```
peer lifecycle chaincode package agrifood_1.tar.gz \
--path ../chaincode/agrifood \
--lang golang \
--label agrifood_1
```

7. **Set Org1's environment variables for CLI operations:**

```
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_ADDRESS=localhost:7051
export
CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org1.exam
ple.com/users/Admin@org1.example.com/msp
export
CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org1.e
xample.com/peers/peer0.org1.example.com/tls/ca.crt
export FABRIC_CFG_PATH=$PWD/../config
export CORE_PEER_TLS_ENABLED=true
```

8. **Install the chaincode on Org1's peer:**

```
peer lifecycle chaincode install agrifood_1.tar.gz
```

9. **Query the installed packages to get the package ID:**

```
peer lifecycle chaincode queryinstalled
```

10. **Approve the chaincode definition for Org1:**

```
peer lifecycle chaincode approveformyorg \
--channelID mychannel \
--name agrifood \
--version 1.0 \
--package-id
agrifood_1:6e7c894ae7849b284eadcd1c58a7775cefc5602984afce378c8d1c8b540e685f \
--sequence 1 \
--signature-policy "OR('Org1MSP.member','Org2MSP.member','Org3MSP.member')" \
--tls \
-o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
msp/tlscacerts/tlsca.example.com-cert.pem
```

11. **Set Org2's environment variables:**

    ```
    export CORE_PEER_LOCALMSPID="Org2MSP"
    export CORE_PEER_ADDRESS=localhost:9051
    export
    CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org2.exam
    ple.com/users/Admin@org2.example.com/msp
    export
    CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org2.e
    xample.com/peers/peer0.org2.example.com/tls/ca.crt
    ```

12. **Approve chaincode for Org2:**

    ```
    peer lifecycle chaincode approveformyorg \
    --channelID mychannel \
    --name agrifood \
    --version 1.0 \
    --package-id
    agrifood_1:6e7c894ae7849b284eadcd1c58a7775cefc5602984afce378c8d1c8b540e685f \
    --sequence 1 \
    --signature-policy "OR('Org1MSP.member','Org2MSP.member','Org3MSP.member')" \
    --tls \
    -o localhost:7050 \
    --ordererTLSHostnameOverride orderer.example.com \
    --cafile
    $PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
    msp/tlscacerts/tlsca.example.com-cert.pem
    ```

13. **Set Org3's environment variables:**

    ```
    export CORE_PEER_LOCALMSPID="Org3MSP"
    export CORE_PEER_ADDRESS=localhost:11051
    export
    CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org3.exam
    ple.com/users/Admin@org3.example.com/msp
    export
    CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org3.e
    xample.com/peers/peer0.org3.example.com/tls/ca.crt
    ```

14. **Approve chaincode for Org3:**

    ```
    peer lifecycle chaincode approveformyorg \
    ```

```
--channelID mychannel \
--name agrifood \
--version 1.0 \
--package-id
agrifood_1:a5b8b5aa9bdcce4c62ab17bb439e45135377827ea2488585641ed9c2700c5252
\
--sequence 1 \
--signature-policy "OR('Org1MSP.member','Org2MSP.member','Org3MSP.member')" \
--tls \
-o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
msp/tlscacerts/tlsca.example.com-cert.pem
```

15. **Commit the chaincode definition (using Org1's environment):**

```
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_ADDRESS=localhost:7051
export
CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org1.exam
ple.com/users/Admin@org1.example.com/msp
export
CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org1.e
xample.com/peers/peer0.org1.example.com/tls/ca.crt
peer lifecycle chaincode commit \
--channelID mychannel \
--name agrifood \
--version 1.0 \
--sequence 1 \
--signature-policy "OR('Org1MSP.member','Org2MSP.member','Org3MSP.member')" \
--tls \
-o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
msp/tlscacerts/tlsca.example.com-cert.pem \
--peerAddresses localhost:7051 \
--tlsRootCertFiles
$PWD/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.co
m/tls/ca.crt \
--peerAddresses localhost:9051 \
--tlsRootCertFiles
$PWD/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.co
m/tls/ca.crt \
--peerAddresses localhost:11051 \
--tlsRootCertFiles
$PWD/organizations/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt
```

Phase 3: Testing Chaincode Functions and Access Control

These commands test the core functionality and validate the access control logic of your chaincode.

16. **Test Case 1: Create a product as Org1 (Farmer):**

    ```
    peer chaincode invoke -o localhost:7050 \
    --ordererTLSHostnameOverride orderer.example.com \
    --tls \
    --cafile
    $PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
    -C mychannel \
    -n agrifood \
    --peerAddresses localhost:7051 \
    --tlsRootCertFiles
    $PWD/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \
    -c '{"function":"CreateProduct","Args":["P003", "Farm C", "Grade C"]}'
    ```

17. **Test Case 2: Query the created product:**

    ```
    peer chaincode query \
    -C mychannel \
    -n agrifood \
    -c '{"function":"GetProduct","Args":["P003"]}'
    ```

18. **Test Case 3: Attempt to create a product as Org2 (Retailer):**

    ```
    export CORE_PEER_LOCALMSPID="Org2MSP"
    export CORE_PEER_ADDRESS=localhost:9051
    export
    CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
    export
    CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
    peer chaincode invoke \
    -o localhost:7050 \
    --ordererTLSHostnameOverride orderer.example.com \
    ```

```
--tls \
--cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
msp/tlscacerts/tlsca.example.com-cert.pem \
-C mychannel \
-n agrifood \
--peerAddresses localhost:9051 \
--tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE \
-c '{"function":"CreateProduct","Args":["P888", "CityMarket", "Grade B"]}'
```

19. **Test Case 4: Update product status as Org2 (Retailer):** *(Ensure Org2's environment variables are set from step 18.)*

```
peer chaincode invoke \
-o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--tls \
--cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
msp/tlscacerts/tlsca.example.com-cert.pem \
-C mychannel \
-n agrifood \
--peerAddresses localhost:9051 \
--tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE \
-c '{"function":"UpdateProductStatus","Args":["P003", "SHIPPED", "Warehouse"]}'
```

20. **Test Case 5: Attempt to update status as Org3 (Customer):**

```
export CORE_PEER_LOCALMSPID="Org3MSP"
export CORE_PEER_ADDRESS=localhost:11051
export
CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org3.exam
ple.com/users/Admin@org3.example.com/msp
export
CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org3.e
xample.com/peers/peer0.org3.example.com/tls/ca.crt
peer chaincode invoke \
-o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--tls \
--cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
msp/tlscacerts/tlsca.example.com-cert.pem \
-C mychannel \
-n agrifood \
--peerAddresses localhost:11051 \
--tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE \
-c '{"function":"UpdateProductStatus","Args":["P003", "IN_TRANSIT", "Org3
Warehouse"]}'
```

21. **Test Case 6: Invoke PurchaseProduct as Org3 (Customer):** *(Ensure Org3's environment variables are set from step 20.)*

    ```
    peer chaincode invoke \
    -o localhost:7050 \
    --ordererTLSHostnameOverride orderer.example.com \
    --tls \
    --cafile
    $PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/
    msp/tlscacerts/tlsca.example.com-cert.pem \
    -C mychannel \
    -n agrifood \
    --peerAddresses localhost:11051 \
    --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE \
    -c '{"function":"PurchaseProduct","Args":["P003"]}
    ```

HYPERLEDGER EXPLORER SETUP

This appendix details the step-by-step process for setting up and configuring Hyperledger Explorer to visualize the Agri-food Supply Chain network.

Phase 1: Explorer Environment Setup

1. **Create a new directory** named explorer and navigate into it.

   ```
   mkdir explorer
   cd explorer
   ```

2. **Copy the necessary configuration files** and the Docker Compose file from the Hyperledger Explorer repository.

   ```
   wget https://raw.githubusercontent.com/hyperledger/blockchain-explorer/main/examples/net1/config.json
   wget https://raw.githubusercontent.com/hyperledger/blockchain-explorer/main/examples/net1/connection-profile/test-network.json -P connection-profile
   wget https://raw.githubusercontent.com/hyperledger/blockchain-explorer/main/docker-compose.yaml
   ```

3. **Copy the cryptographic material** from your Fabric network to the explorer directory. This is essential for the Explorer to authenticate with the network. If you encounter permission errors, use sudo chown to fix file ownership before copying.

   ```
   # Fix file ownership (if needed)
   sudo chown -R abdullah:abdullah /home/abdullah/go/src/github.com/Agrifood/fabric-samples/test-network/organizations

   # Copy the organizations directory
   cp -r /home/abdullah/go/src/github.com/Agrifood/fabric-samples/test-network/organizations \
   /home/abdullah/explorer/
   ```

Phase 2: Configuration and Launch

1. **Edit docker-compose.yaml** to connect to your running network. You must set the external network name to fabric_test and update the volume path to your local crypto material.

   YAML

   ```
   # ...
   networks:
     mynetwork.com:
       external: true
   ```

```
    name: fabric_test
# ...
services:
  explorer.mynetwork.com:
    # ...
    volumes:
      - ./config.json:/opt/explorer/app/platform/fabric/config.json
      - ./connection-profile:/opt/explorer/app/platform/fabric/connection-profile
      - /home/abdullah/go/src/github.com/Agrifood/fabric-samples/test-network/organizations:/tmp/crypto
      - walletstore:/opt/explorer/wallet
# ...
```

2. **Edit test-network.json** to specify the correct admin private key and signed certificate paths for Org1MSP.

   JSON

   ```
   "organizations": {
      "Org1MSP": {
         "mspid": "Org1MSP",
         "adminPrivateKey": {
            "path": "/tmp/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/YOUR_PRIVATE_KEY_HERE_sk"
         },
         "peers": ["peer0.org1.example.com"],
         "signedCert": {
            "path": "/tmp/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/signcerts/cert.pem"
         }
      }
   },
   // ...
   ```

3. **Launch the Explorer services** after ensuring your Fabric network is running.

   ```
   docker-compose up –d
   ```

Phase 3: Verification and Debugging

1. **Access the Dashboard:** Open a web browser and navigate to http://localhost:8080.
2. **Check Logs:** If the Explorer fails to start, you can view the logs for detailed error messages.

   ```
   docker logs -f explorer.mynetwork.com
   ```

This setup provides a complete visual tool to monitor the blocks and transactions of your Agri-food supply chain, which is essential for validating your system.