Assignment 2

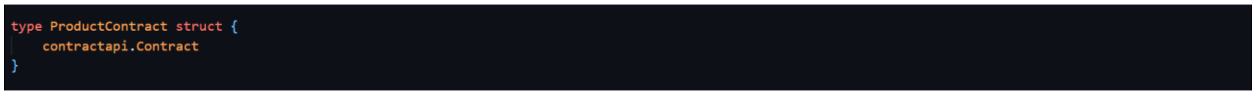
GCIT

Yeshi Chogyel/Blockchain/Third Year

Sonam Tshering

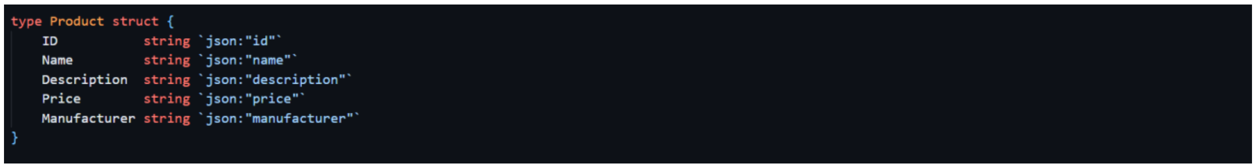
5/20/2024

1.Chain Code



```go
type ProductContract struct {
    contractapi.Contract
}
```

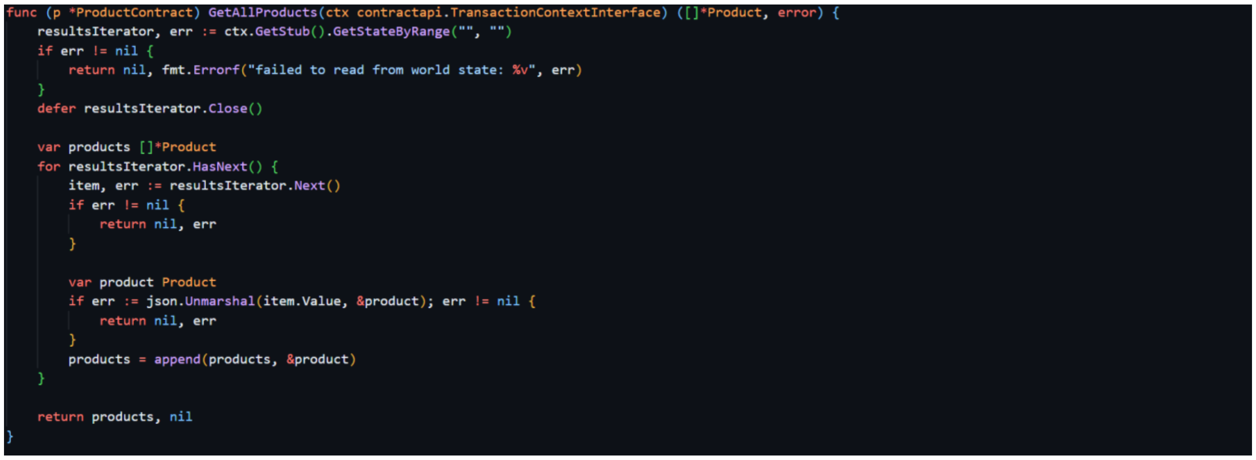
- ProductContract is defined as struct and it will contain methods related to product information in the blockchain and it inherits all the properties of the contractapi.contract.



```go
type Product struct {
    ID           string `json:"id"`
    Name         string `json:"name"`
    Description  string `json:"description"`
    Price        string `json:"price"`
    Manufacturer string `json:"manufacturer"`
}
```

- This Product struct defines data model for the product that is ID, Name, Description, Price, and Manufacturer.

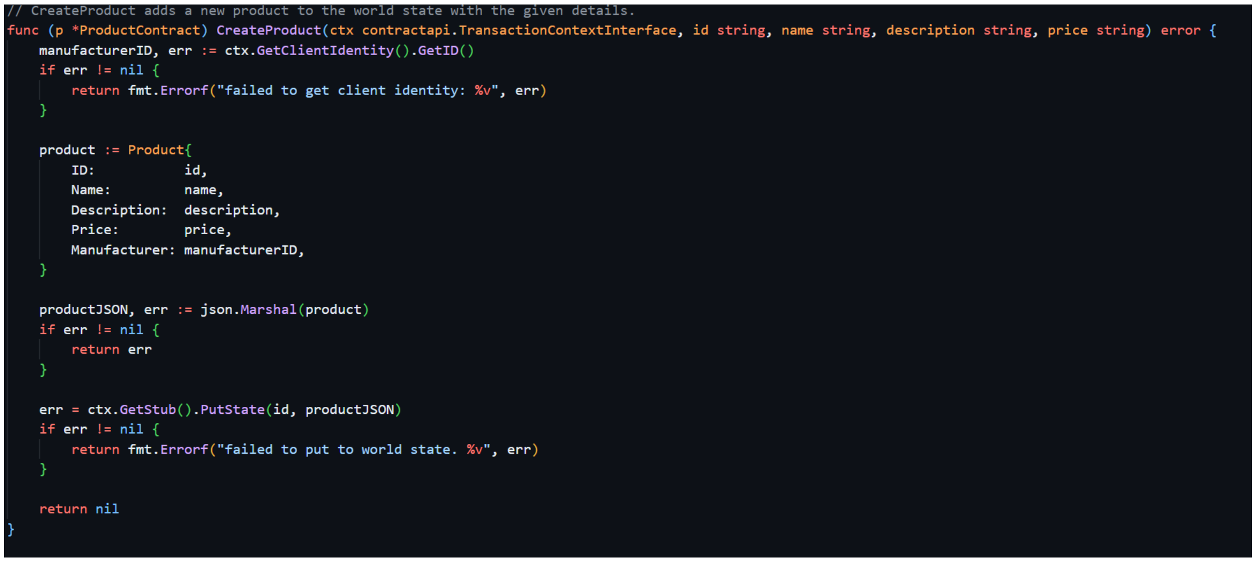
GetAllProducts Method



```go
func (p *ProductContract) GetAllProducts(ctx contractapi.TransactionContextInterface) ([]*Product, error) {
    resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
    if err != nil {
        return nil, fmt.Errorf("failed to read from world state: %v", err)
    }
    defer resultsIterator.Close()

    var products []*Product
    for resultsIterator.HasNext() {
        item, err := resultsIterator.Next()
        if err != nil {
            return nil, err
        }

        var product Product
        if err := json.Unmarshal(item.Value, &product); err != nil {
            return nil, err
        }
        products = append(products, &product)
    }

    return products, nil
}
```

- This function retirves all the data stored in the world state.
- The function uses GetStateByRange("", "") to fetch all key-value pairs and iterates over the result and appends them to list.

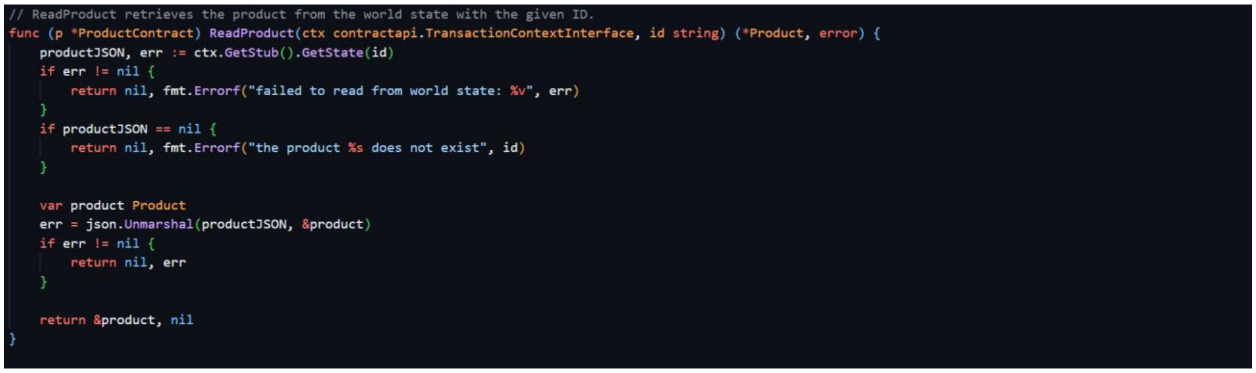
CreateProduct Method



```go
// CreateProduct adds a new product to the world state with the given details.
func (p *ProductContract) CreateProduct(ctx contractapi.TransactionContextInterface, id string, name string, description string, price string) error {
    manufacturerID, err := ctx.GetClientIdentity().GetID()
    if err != nil {
        return fmt.Errorf("failed to get client identity: %v", err)
    }

    product := Product{
        ID:           id,
        Name:         name,
        Description:  description,
        Price:        price,
        Manufacturer: manufacturerID,
    }

    productJSON, err := json.Marshal(product)
    if err != nil {
        return err
    }

    err = ctx.GetStub().PutState(id, productJSON)
    if err != nil {
        return fmt.Errorf("failed to put to world state. %v", err)
    }

    return nil
}
```

- This function creates new product and stores in the world state and retrieves the manufacturer from the client identity.

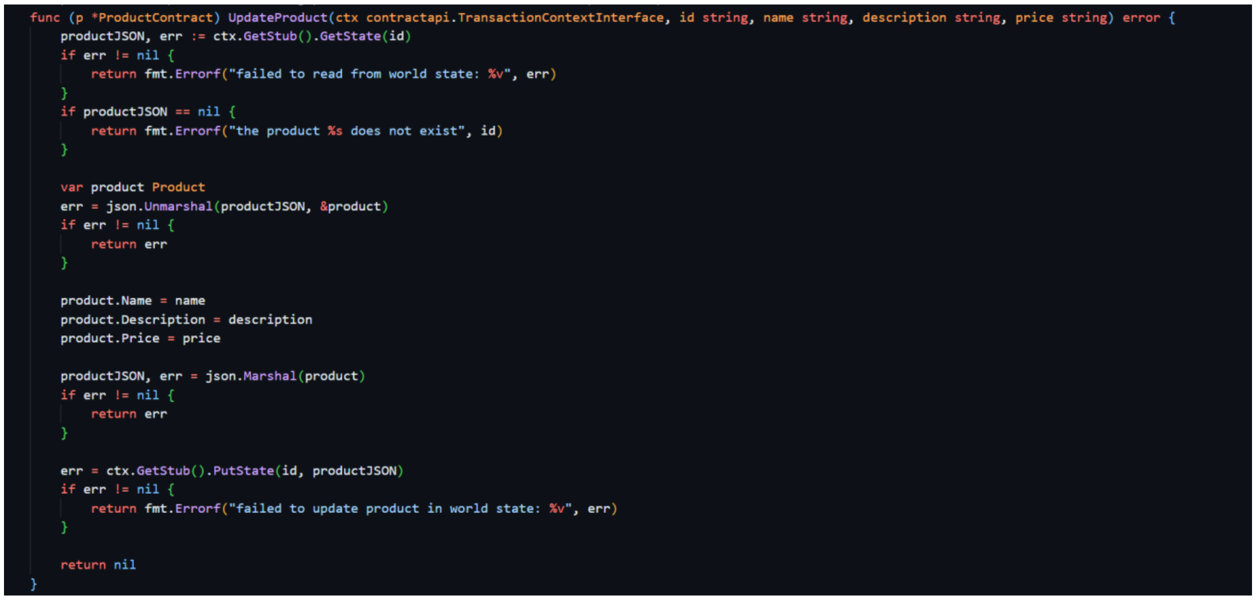
ReadProduct Method



```go
// ReadProduct retrieves the product from the world state with the given ID.
func (p *ProductContract) ReadProduct(ctx contractapi.TransactionContextInterface, id string) (*Product, error) {
    productJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return nil, fmt.Errorf("failed to read from world state: %v", err)
    }
    if productJSON == nil {
        return nil, fmt.Errorf("the product %s does not exist", id)
    }

    var product Product
    err = json.Unmarshal(productJSON, &product)
    if err != nil {
        return nil, err
    }

    return &product, nil
}
```

- This function retrives a product bu ID from the world state and uses “GetState” to fetch data and returns the data.

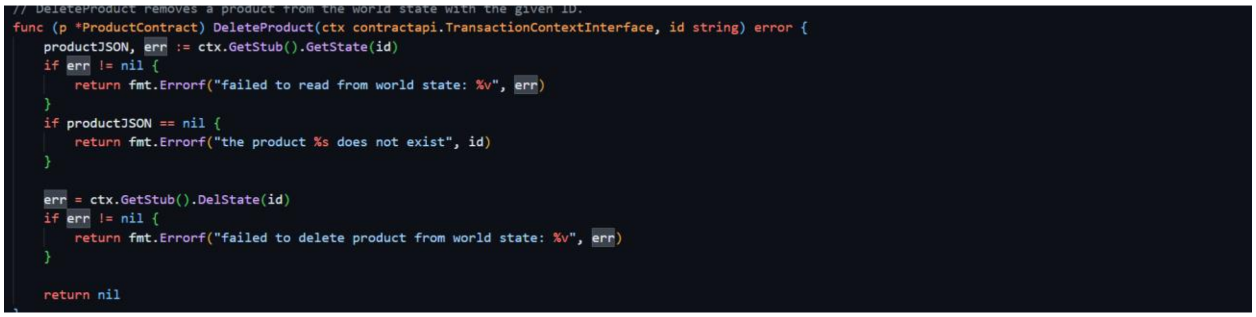
UpdateProduct Method



```go
func (p *ProductContract) UpdateProduct(ctx contractapi.TransactionContextInterface, id string, name string, description string, price string) error {
    productJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return fmt.Errorf("failed to read from world state: %v", err)
    }
    if productJSON == nil {
        return fmt.Errorf("the product %s does not exist", id)
    }

    var product Product
    err = json.Unmarshal(productJSON, &product)
    if err != nil {
        return err
    }

    product.Name = name
    product.Description = description
    product.Price = price

    productJSON, err = json.Marshal(product)
    if err != nil {
        return err
    }

    err = ctx.GetStub().PutState(id, productJSON)
    if err != nil {
        return fmt.Errorf("failed to update product in world state: %v", err)
    }

    return nil
}
```

- This method will update the existing products details. It will fetch the product by ID updates its field and saves it back to the world state.

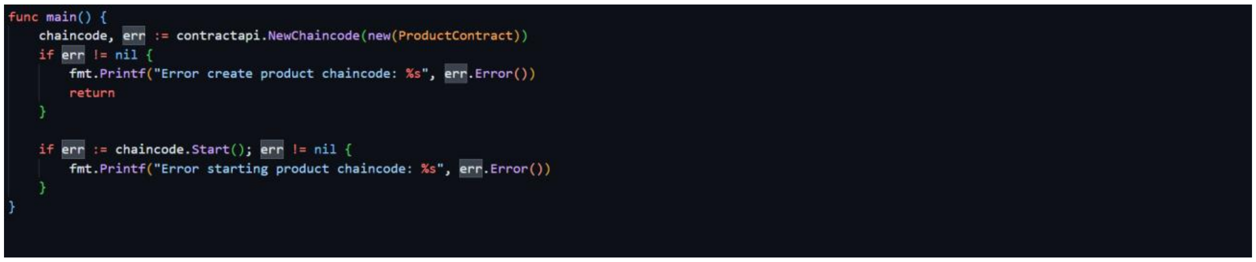
DeleteProduct



```go
// DeleteProduct removes a product from the world state with the given ID.
func (p *ProductContract) DeleteProduct(ctx contractapi.TransactionContextInterface, id string) error {
    productJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return fmt.Errorf("failed to read from world state: %v", err)
    }
    if productJSON == nil {
        return fmt.Errorf("the product %s does not exist", id)
    }

    err = ctx.GetStub().DelState(id)
    if err != nil {
        return fmt.Errorf("failed to delete product from world state: %v", err)
    }

    return nil
```

- This method will delete a product using the ID from the world state and uses DelState to remove the product data.

MainMethod

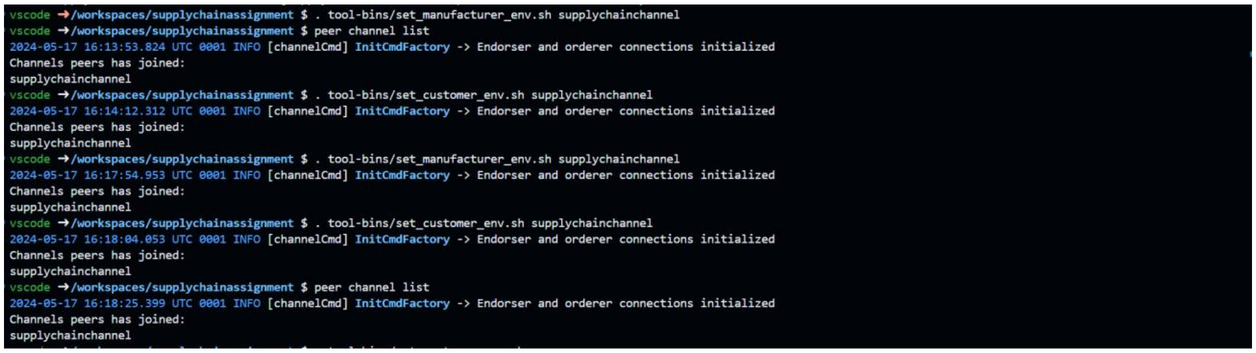


```go
func main() {
    chaincode, err := contractapi.NewChaincode(new(ProductContract))
    if err != nil {
        fmt.Printf("Error create product chaincode: %s", err.Error())
        return
    }

    if err := chaincode.Start(); err != nil {
        fmt.Printf("Error starting product chaincode: %s", err.Error())
    }
}
```

- This is the main function of the chaincode application and it creates a new chain code instance with the ProductContract and starts it.

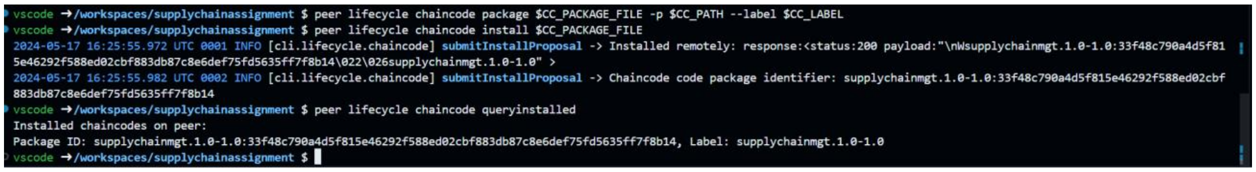
2.Compiling and Packaging the Chaincode

Set the environment variable for the packaging in set_custoemer_env.sh and set_manufacture_env.sh and run that file:



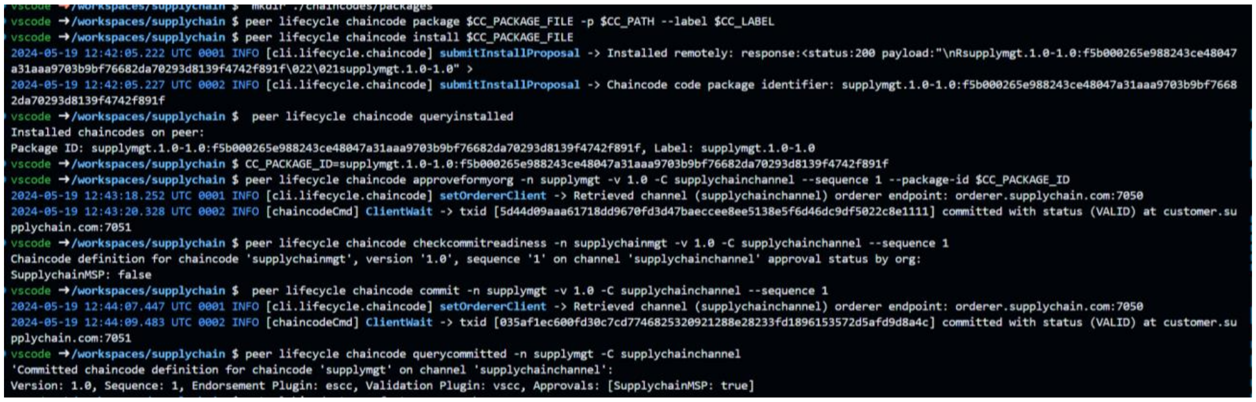
```
vscode →/workspaces/supplychainassignment $ . tool-bins/set_manufacturer_env.sh supplychainchannel
vscode →/workspaces/supplychainassignment $ peer channel list
2024-05-17 16:13:53.824 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
Channels peers has joined:
supplychainchannel
vscode →/workspaces/supplychainassignment $ . tool-bins/set_customer_env.sh supplychainchannel
2024-05-17 16:14:12.312 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
Channels peers has joined:
supplychainchannel
vscode →/workspaces/supplychainassignment $ . tool-bins/set_manufacturer_env.sh supplychainchannel
2024-05-17 16:17:54.953 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
Channels peers has joined:
supplychainchannel
vscode →/workspaces/supplychainassignment $ . tool-bins/set_customer_env.sh supplychainchannel
2024-05-17 16:18:04.053 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
Channels peers has joined:
supplychainchannel
vscode →/workspaces/supplychainassignment $ peer channel list
2024-05-17 16:18:25.399 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
Channels peers has joined:
supplychainchannel
```

After that package the chain code and install it using the followinf command:



```
vscode →/workspaces/supplychainassignment $ peer lifecycle chaincode package $CC_PACKAGE_FILE -p $CC_PATH --label $CC_LABEL
vscode →/workspaces/supplychainassignment $ peer lifecycle chaincode install $CC_PACKAGE_FILE
2024-05-17 16:25:55.972 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nWsupplychainmgt.1.0-1.0:33f48c790a4d5f81
5e46292f588ed02cbf883db87c8e6def75fd5635ff7f8b14\022\026supplychainmgt.1.0-1.0" >
2024-05-17 16:25:55.982 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: supplychainmgt.1.0-1.0:33f48c790a4d5f815e46292f588ed02cbf
883db87c8e6def75fd5635ff7f8b14
vscode →/workspaces/supplychainassignment $ peer lifecycle chaincode queryinstalled
Installed chaincodes on peer:
Package ID: supplychainmgt.1.0-1.0:33f48c790a4d5f815e46292f588ed02cbf883db87c8e6def75fd5635ff7f8b14, Label: supplychainmgt.1.0-1.0
vscode →/workspaces/supplychainassignment $
```

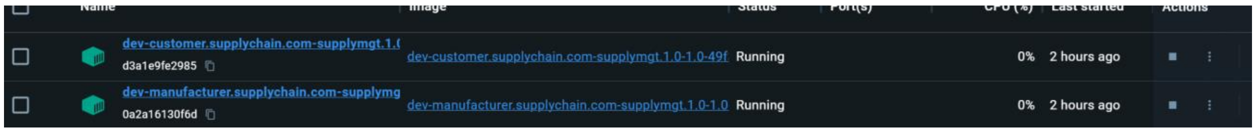
Approve, commit and verify the committed chain code:



```
vscode →/workspaces/supplychain $ mkdir ./chaincodes/packages
vscode →/workspaces/supplychain $ peer lifecycle chaincode package $CC_PACKAGE_FILE -p $CC_PATH --label $CC_LABEL
vscode →/workspaces/supplychain $ peer lifecycle chaincode install $CC_PACKAGE_FILE
2024-05-19 12:42:05.222 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nRsupplymgt.1.0-1.0:f5b000265e988243ce48047
a31aaa9703b9bf76682da70293d8139f4742f891f\022\021supplymgt.1.0-1.0" >
2024-05-19 12:42:05.227 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: supplymgt.1.0-1.0:f5b000265e988243ce48047a31aaa9703b9bf7668
2da70293d8139f4742f891f
vscode →/workspaces/supplychain $  peer lifecycle chaincode queryinstalled
Installed chaincodes on peer:
Package ID: supplymgt.1.0-1.0:f5b000265e988243ce48047a31aaa9703b9bf76682da70293d8139f4742f891f, Label: supplymgt.1.0-1.0
vscode →/workspaces/supplychain $ CC_PACKAGE_ID=supplymgt.1.0-1.0:f5b000265e988243ce48047a31aaa9703b9bf76682da70293d8139f4742f891f
vscode →/workspaces/supplychain $ peer lifecycle chaincode approveformyorg -n supplymgt -v 1.0 -C supplychainchannel --sequence 1 --package-id $CC_PACKAGE_ID
2024-05-19 12:43:18.252 UTC 0001 INFO [cli.lifecycle.chaincode] setOrdererClient -> Retrieved channel (supplychainchannel) orderer endpoint: orderer.supplychain.com:7050
2024-05-19 12:43:20.328 UTC 0002 INFO [chaincodeCmd] ClientWait -> txid [5d44d09aaa61718dd9670fd3d47baeccee8ee5138e5f6d46dc9df5022c8e1111] committed with status (VALID) at customer.su
pplychain.com:7051
vscode →/workspaces/supplychain $ peer lifecycle chaincode checkcommitreadiness -n supplychainmgt -v 1.0 -C supplychainchannel --sequence 1
Chaincode definition for chaincode 'supplychainmgt', version '1.0', sequence '1' on channel 'supplychainchannel' approval status by org:
SupplychainMSP: false
vscode →/workspaces/supplychain $  peer lifecycle chaincode commit -n supplymgt -v 1.0 -C supplychainchannel --sequence 1
2024-05-19 12:44:07.447 UTC 0001 INFO [cli.lifecycle.chaincode] setOrdererClient -> Retrieved channel (supplychainchannel) orderer endpoint: orderer.supplychain.com:7050
2024-05-19 12:44:09.483 UTC 0002 INFO [chaincodeCmd] ClientWait -> txid [035af1ec600fd30c7cd774682532092128 8e28233fd1896153572d5afd9d8a4c] committed with status (VALID) at customer.su
pplychain.com:7051
vscode →/workspaces/supplychain $ peer lifecycle chaincode querycommitted -n supplymgt -C supplychainchannel
'Committed chaincode definition for chaincode 'supplymgt' on channel 'supplychainchannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [SupplychainMSP: true]
```

We need to do that for all the other organisation.

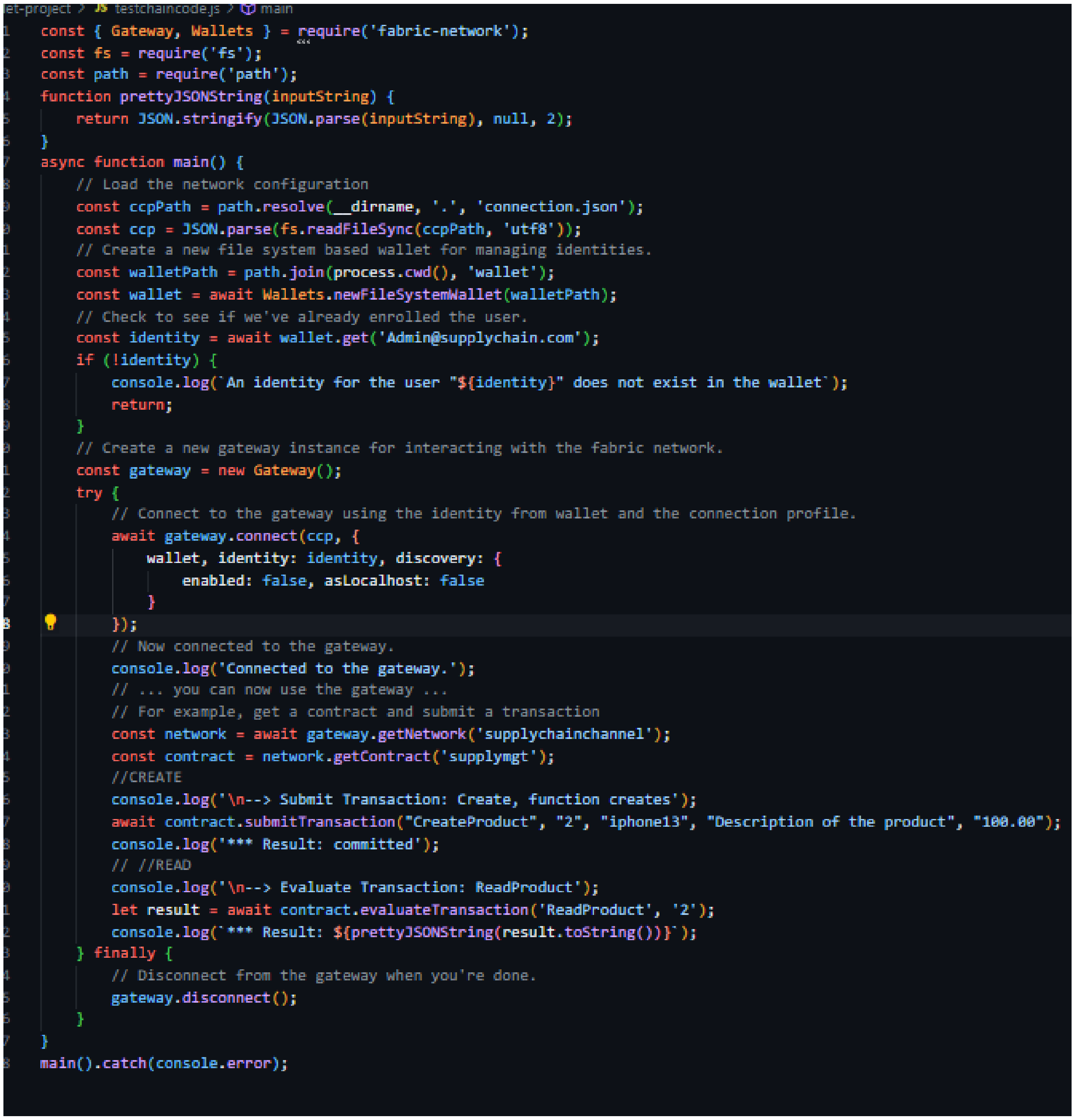
After completing this step, container of the orgaisation will be running in the docker:



	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
☐	dev-customer.supplychain.com-supplymgt.1.(d3a1e9fe2985	dev-customer.supplychain.com-supplymgt.1.0-1.0-49f	Running		0%	2 hours ago	■ ⋮
☐	dev-manufacturer.supplychain.com-supplymg 0a2a16130f6d	dev-manufacturer.supplychain.com-supplymgt.1.0-1.0	Running		0%	2 hours ago	■ ⋮

3.Testing chain code

To test the chain code testchaincode.js was created:



```
const { Gateway, Wallets } = require('fabric-network');
const fs = require('fs');
const path = require('path');
function prettyJSONString(inputString) {
    return JSON.stringify(JSON.parse(inputString), null, 2);
}
async function main() {
    // Load the network configuration
    const ccpPath = path.resolve(__dirname, '.', 'connection.json');
    const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'));
    // Create a new file system based wallet for managing identities.
    const walletPath = path.join(process.cwd(), 'wallet');
    const wallet = await Wallets.newFileSystemWallet(walletPath);
    // Check to see if we've already enrolled the user.
    const identity = await wallet.get('Admin@supplychain.com');
    if (!identity) {
        console.log(`An identity for the user "${identity}" does not exist in the wallet`);
        return;
    }
    // Create a new gateway instance for interacting with the fabric network.
    const gateway = new Gateway();
    try {
        // Connect to the gateway using the identity from wallet and the connection profile.
        await gateway.connect(ccp, {
            wallet, identity: identity, discovery: {
                enabled: false, asLocalhost: false
            }
        });
        // Now connected to the gateway.
        console.log('Connected to the gateway.');
        // ... you can now use the gateway ...
        // For example, get a contract and submit a transaction
        const network = await gateway.getNetwork('supplychainchannel');
        const contract = network.getContract('supplymgt');
        //CREATE
        console.log('\n--> Submit Transaction: Create, function creates');
        await contract.submitTransaction("CreateProduct", "2", "iphone13", "Description of the product", "100.00");
        console.log('*** Result: committed');
        // //READ
        console.log('\n--> Evaluate Transaction: ReadProduct');
        let result = await contract.evaluateTransaction('ReadProduct', '2');
        console.log(`*** Result: ${prettyJSONString(result.toString())}`);
    } finally {
        // Disconnect from the gateway when you're done.
        gateway.disconnect();
    }
}
main().catch(console.error);
```

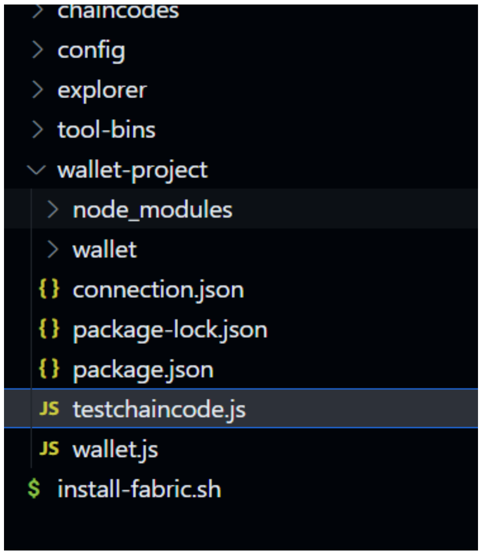
We can test the chain code the terminal by invoking the functions of the chain code:

For example, this command will invoke ReadProducts with id 1:

```
peer chaincode query -C supplychainchannel -n supplytmgt -'{"function":"ReadProducts","Args":["1"]}'
```

4.Creating Wallet

To create the wallet creat a folder like this:



```
> chaincodes
> config
> explorer
> tool-bins
v wallet-project
  > node_modules
  > wallet
  {} connection.json
  {} package-lock.json
  {} package.json
  JS testchaincode.js
  JS wallet.js
$ install-fabric.sh
```

In wallet.js add the following code:

```
const fs = require('fs');
const path = require('path');
const { Wallets, FileSystemWallet, X509WalletMixin } = require('fabric-network');
const CRYPTO_CONFIG = path.resolve(__dirname, '../config/crypto-config');
const CRYPTO_CONFIG_PEER_ORGS = path.join(CRYPTO_CONFIG, 'peerOrganizations')
const WALLET_FOLDER = './wallet'

var wallet
main();
async function main() {
 let action = 'list'
 if (process.argv.length > 2) {
  action = process.argv[2]
 }
 else {
  console.log(
   `Usage: node wallet.js list
    node wallet.js add supplychain Admin)
    Not enough arguments.`)
  return
 }
 console.log(CRYPTO_CONFIG_PEER_ORGS)
 wallet = await Wallets.newFileSystemWallet(WALLET_FOLDER)
 console.log(process.argv.length)
 if (action == 'list') {
  console.log("List of identities in wallet:")
  listIdentities()
 } else if (action == 'add' || action == 'export') {
  if (process.argv.length < 5) {
   console.log("For 'add' & 'export' - Org & User are needed!!!")
   process.exit(1)
  }
  if (action == 'add') {
   addToWallet(process.argv[3], process.argv[4])
   console.log('Done adding/updating.')
  } else {
   exportIdentity(process.argv[3], process.argv[4])
  }
 }
}

/**
 * @param   string
 * @param   string
 */
async function addToWallet(org, user) {
 try {
  var cert = readCertCryptogen(org, user)

  var key = readPrivateKeyCryptogen(org, user)

 } catch (e) {
  console.log("Error reading certificate or key!!! " + org + "/" + user)
  process.exit(1)
 }

 let mspId = createMSPId(org)

 const identityLabel = createIdentityLabel(org, user);

 const userIdentity = await wallet.get(identityLabel);
 if (userIdentity) {
  console.log(`An identity for the user "${identityLabel}" already exists in the wallet`);
  return;
 }
 const identity = {
  credentials: {
   certificate: cert,
      privateKey: key,
    },
    mspId: mspId,
    type: 'X.509',
  };

  await wallet.put(identityLabel, identity);

  console.log(`Successfully added user "${identityLabel}" to the wallet`);
}

async function listIdentities() {
  console.log("Identities in Wallet:")

    const identities = await wallet.list();

    for (const identity of identities) {
        console.log(`user: ${identity}`);
    }

}

/**
* @param {string} org
* @param {string} user
*/
async function exportIdentity(org, user) {
  // Label is used for identifying the identity in wallet
  let label = createIdentityLabel(org, user)

  // To retrive execute export
  let identity = await wallet.export(label)

  if (identity == null) {
    console.log(`Identity ${user} for ${org} Org Not found!!!`)
  } else {
    // Prints all attributes : label, Key, Cert
    console.log(identity)
  }
}

/**
 * Reads content of the certificate
 * @param {string} org
 * @param {string} user
 */
function readCertCryptogen(org, user) {
  //budget.com/users/Admin@budget.com/msp/signcerts/Admin@budget.com-cert.pem"
  var certPath = CRYPTO_CONFIG_PEER_ORGS + "/" + org + ".com/users/" + user + "@" + org +
".com/msp/signcerts/" + user + "@" + org + ".com-cert.pem"
  const cert = fs.readFileSync(certPath).toString();
  return cert
}

/**
 * Reads the content of users private key
 * @param {string} org
 * @param {string} user
 */
function readPrivateKeyCryptogen(org, user) {
  // ../crypto/crypto-
config/peerOrganizations/budget.com/users/Admin@budget.com/msp/keystore/05beac9849f610ad5cc
8997e5f45343ca918de78398988def3f288b60d8ee27c_sk
  var pkFolder = CRYPTO_CONFIG_PEER_ORGS + "/" + org + ".com/users/" + user + "@" + org +
".com/msp/keystore"
  fs.readdirSync(pkFolder).forEach(file => {
    // console.log(file);
    // return the first file
    pkfile = file
    return
  })

  return fs.readFileSync(pkFolder + "/" + pkfile).toString()
}

/**
 * Utility function
 * Creates the MSP ID from the org name for 'acme' it will be 'AcmeMSP'
 * @param {string} org
 */
function createMSPId(org) {
  return org.charAt(0).toUpperCase() + org.slice(1) + 'MSP'
}

/**
 * Utility function
 * Creates an identity label for the wallet
 * @param {string} org
 * @param {string} user
 */
```

```
function createIdentityLabel(org, user) {
  return user + '@' + org + '.com';
}
```

Code explanation:

Importing Dependencies:

- Importing necessary modules such as fs (file system) and path for file access, and {Wallets} from fabric-network for managing the file system wallet.

Configuration Constants:

- Defines CRYPTO_CONFIG and CRYPTO_CONFIG_PEER_ORGS constants specifying the location of cryptographic material and peer organization configurations, respectively.

Wallet Folder:

- Wallet identities are stored in a folder named ./wallet.

Main Function:

- Determines the requested action (e.g., list, add, or export) based on command-line arguments.
- Creates a wallet instance using Wallets.newFileSystemWallet(WALLET_FOLDER).

Actions:

- Handles actions like listing identities, adding or updating an identity, and exporting an identity.

addToWallet(org, user) Function:

- Reads certificate and key content, constructs MSP ID, generates a unique identity label, and adds the user's identity to the wallet.

listIdentities() Function:

- Asynchronously lists and prints the identities stored in the wallet.

exportIdentity(org, user) Function:

- Generates a unique identity label, retrieves the identity associated with the label from the wallet, and prints its attributes.

readCertCryptogen(org, user) Function:

- Reads the content of the certificate file for a given organization and user.

readPrivateKeyCryptogen(org, user) Function:

- Reads the content of a user's private key file.

createMSPId(org) Function:

- Creates the MSP ID from the organization name.

createIdentityLabel(org, user) Function:

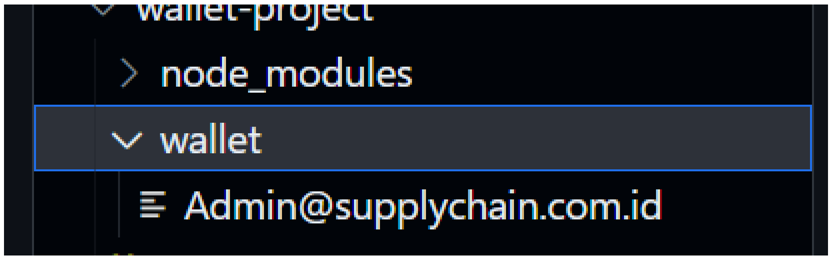
- Generates an identity label for the wallet in the format: user@org.com.

5.Run the Wallet Application

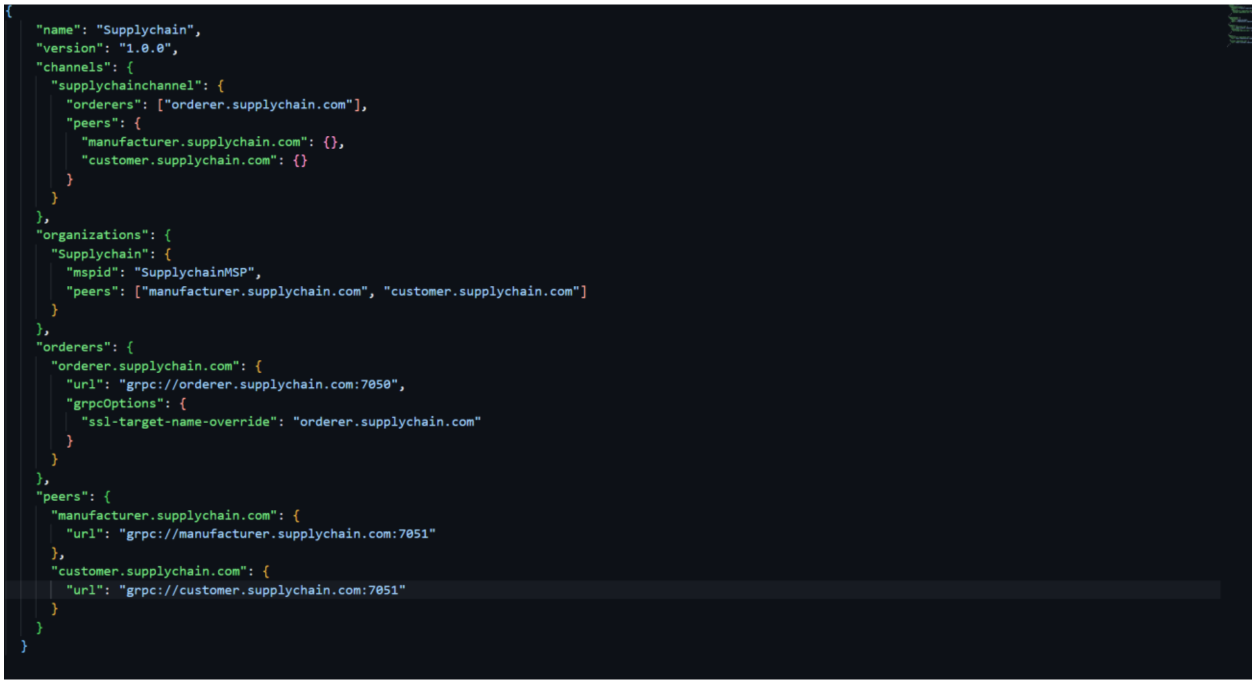
Use this command to start the wallet.js (wallet applicayion).

- Node wallect.js

After running that we will get this:



6.Network Connection Profile



```
{
    "name": "Supplychain",
    "version": "1.0.0",
    "channels": {
      "supplychainchannel": {
        "orderers": ["orderer.supplychain.com"],
        "peers": {
          "manufacturer.supplychain.com": {},
          "customer.supplychain.com": {}
        }
      }
    },
    "organizations": {
      "Supplychain": {
        "mspid": "SupplychainMSP",
        "peers": ["manufacturer.supplychain.com", "customer.supplychain.com"]
      }
    },
    "orderers": {
      "orderer.supplychain.com": {
        "url": "grpc://orderer.supplychain.com:7050",
        "grpcOptions": {
          "ssl-target-name-override": "orderer.supplychain.com"
        }
      }
    },
    "peers": {
      "manufacturer.supplychain.com": {
        "url": "grpc://manufacturer.supplychain.com:7051"
      },
      "customer.supplychain.com": {
        "url": "grpc://customer.supplychain.com:7051"
      }
    }
  }
```

Network Information: Addresses of the network's peers, orderers, and certificate authorities.

Channel Information: Names of the channels that the client can interact with.

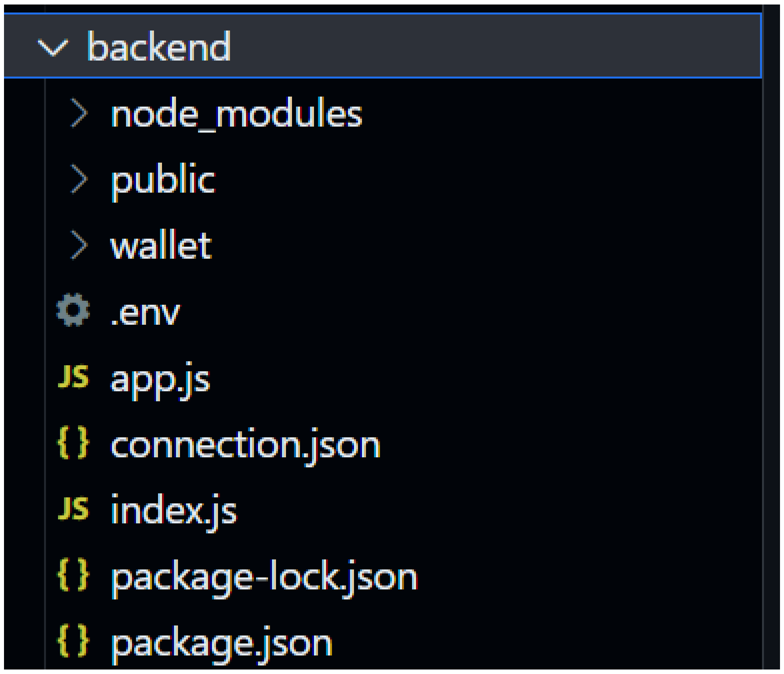
Organizations: Information about the organizations participating in the network, including MSP (Membership Service Provider) details.

Certificate Authorities: Details about the network's certificate authorities that issue and manage digital certificates.

TLS Settings: Configuration related to Transport Layer Security (TLS) for secure communication.

User Credentials: Information about the user or admin credentials that the client will use to authenticate with the network. When the gateway.connect method is called, it uses this file to establish a connection to the Fabric network, allowing the client application to submit transactions, query the ledger, and listen for events. The file is crucial for setting up the client-side of a Fabric application to ensure it can communicate correctly with the blockchain components. It's often manually created by a network Administrator.

7.Backend



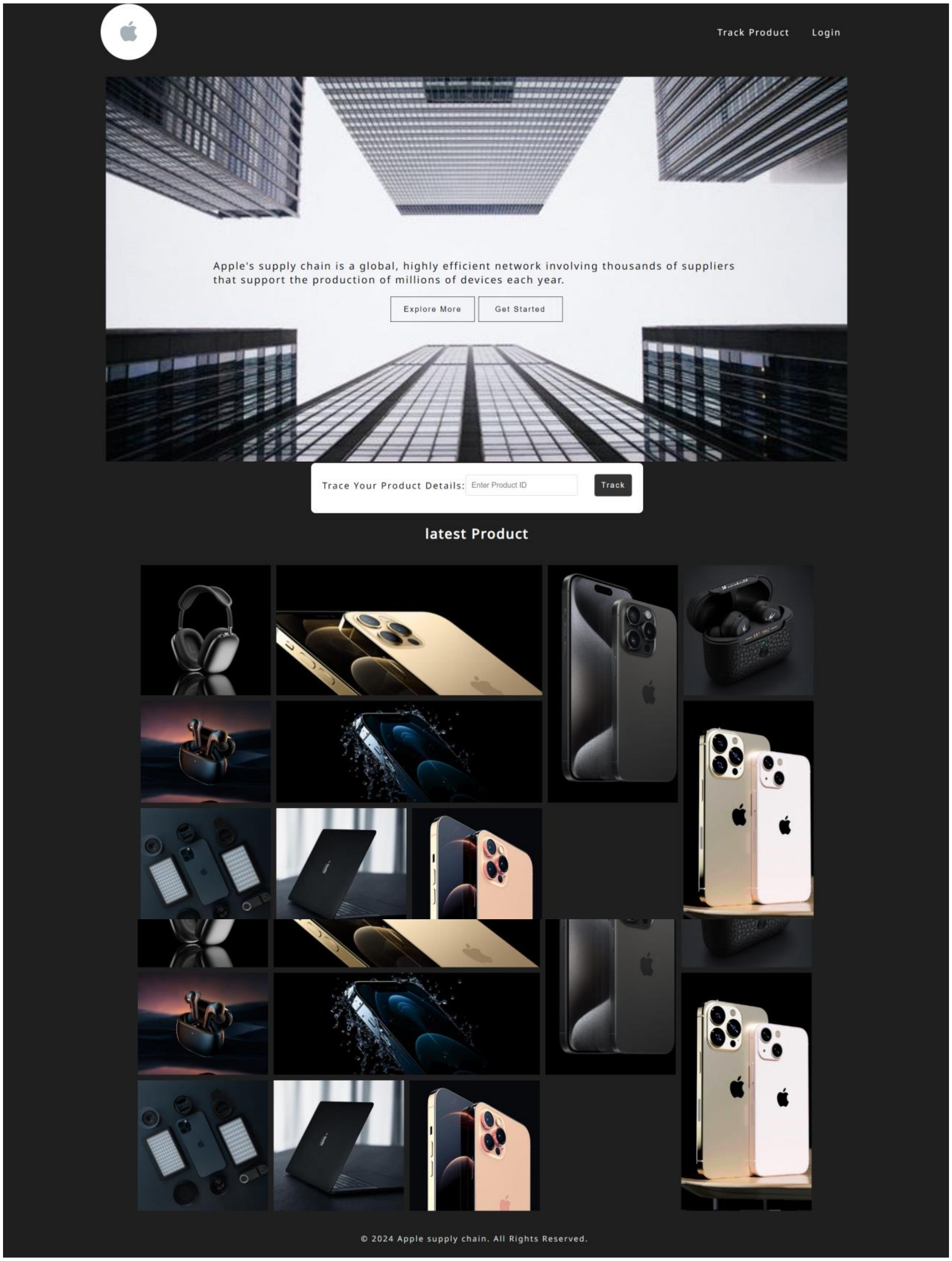
```
backend
  node_modules
  public
  wallet
  .env
  app.js
  connection.json
  index.js
  package-lock.json
  package.json
```

In this folder we set up the RESTful web services which is the app.js.

Within RESTful api we create d Wallets classes from fabric-network would be used within these functions to connect to the blockchain network and submit or evaluate transactions. The fs and path modules are used to handle file system operations, likely for reading the connection profile and wallet path. After that are all created, we test them using postman. After successfully testing the RESTful api we integrate the UI and the chain code.

8.Workingof the application

Customer:



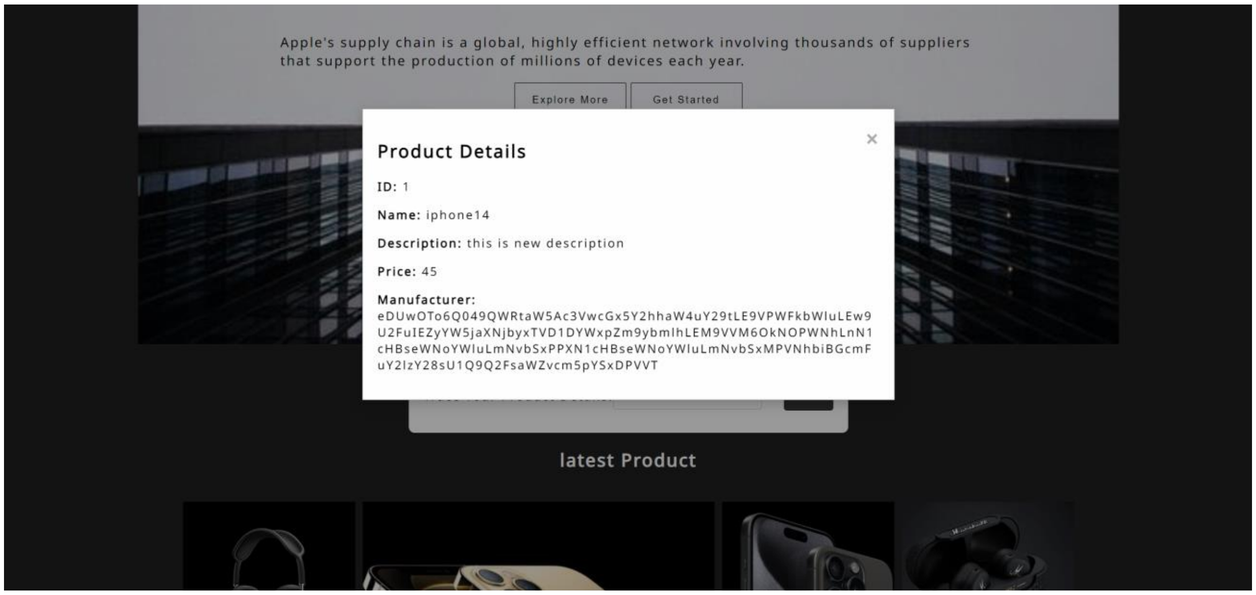
Apple's supply chain is a global, highly efficient network involving thousands of suppliers that support the production of millions of devices each year.

latest Product

© 2024 Apple supply chain. All Rights Reserved.

As the application starts the user will be re directed to this landing page where a user can track the product details using the product id given by the manufacturer.

When user give the correct product Id they will be provided with the product details:



Apple's supply chain is a global, highly efficient network involving thousands of suppliers that support the production of millions of devices each year.

Product Details

ID: 1

Name: iphone14

Description: this is new description

Price: 45

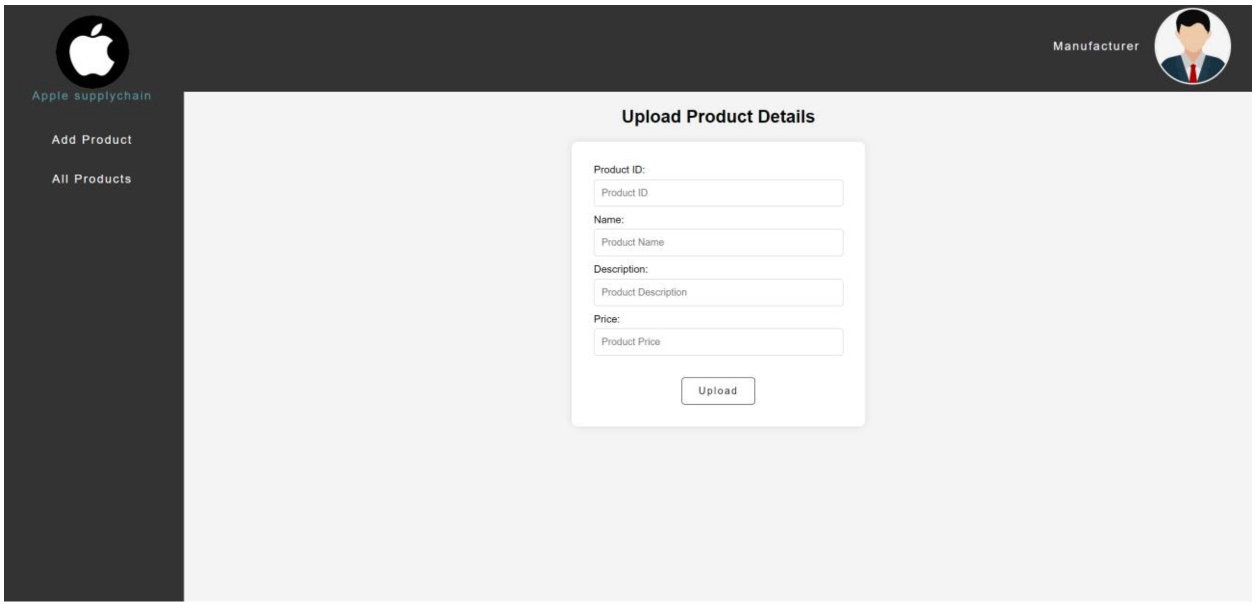
Manufacturer:
eDUwOTo6Q049QWRtaW5Ac3VwcGx5Y2hhaW4uY29tLE9VPWFkbWluLEw9U2FuIEZyYW5jaXNjbyxTVD1DYWxpZm9ybmlhLEM9VVM6OkNOPWNhLnN1cHBseWNoYWluLmNvbSxPPXN1cHBseWNoYWluLmNvbSxMPVNhbiBGcmFuY2lzY28sU1Q9Q2FsaWZvcm5pYSxDPVVT

By cross-referencing the product ID with the manufacturer ID stored in the blockchain, users can ensure that the product they have is genuine and not counterfeit. This approach leverages the immutable and transparent nature of blockchain to provide a reliable verification mechanism. It's a clever use of technology to combat counterfeiting.

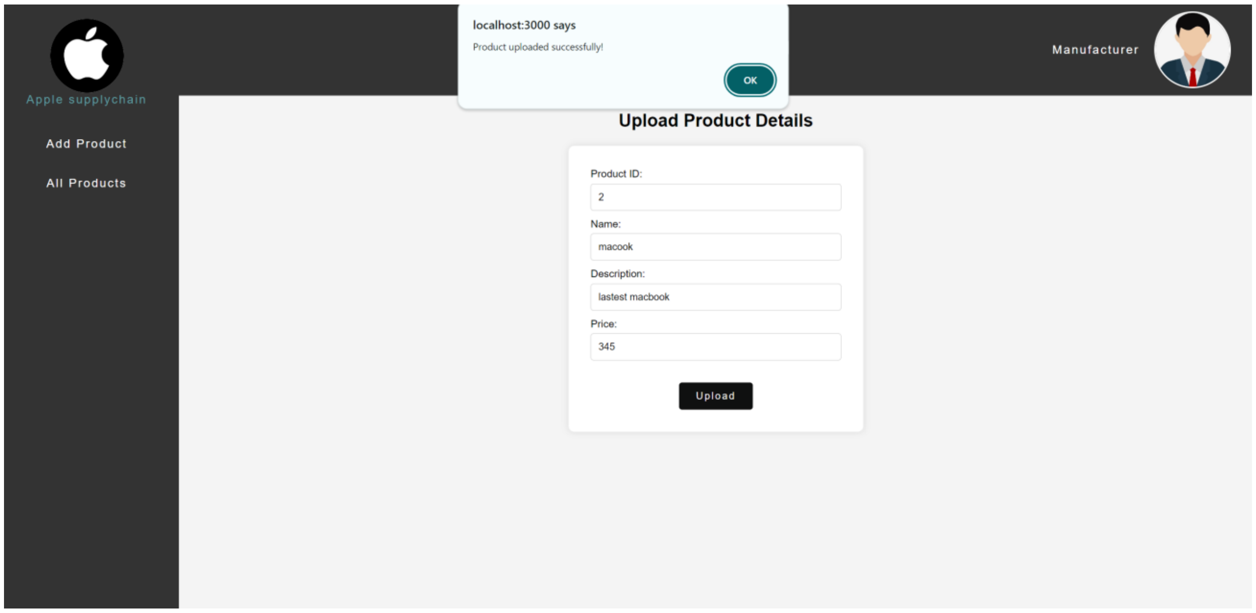
Manufacturer:

With the ability for manufacturers to add, delete, and update product details stored in the ledger, it ensures that the information remains up-to-date and accurate. This adds another layer of trust for consumers, knowing that the information they're accessing is current and directly from the manufacturer.

The manufacturer will be redirected to this dashboard upon staring the system:



Manufacturer will add the product details form this form which will be stored in the ledger:

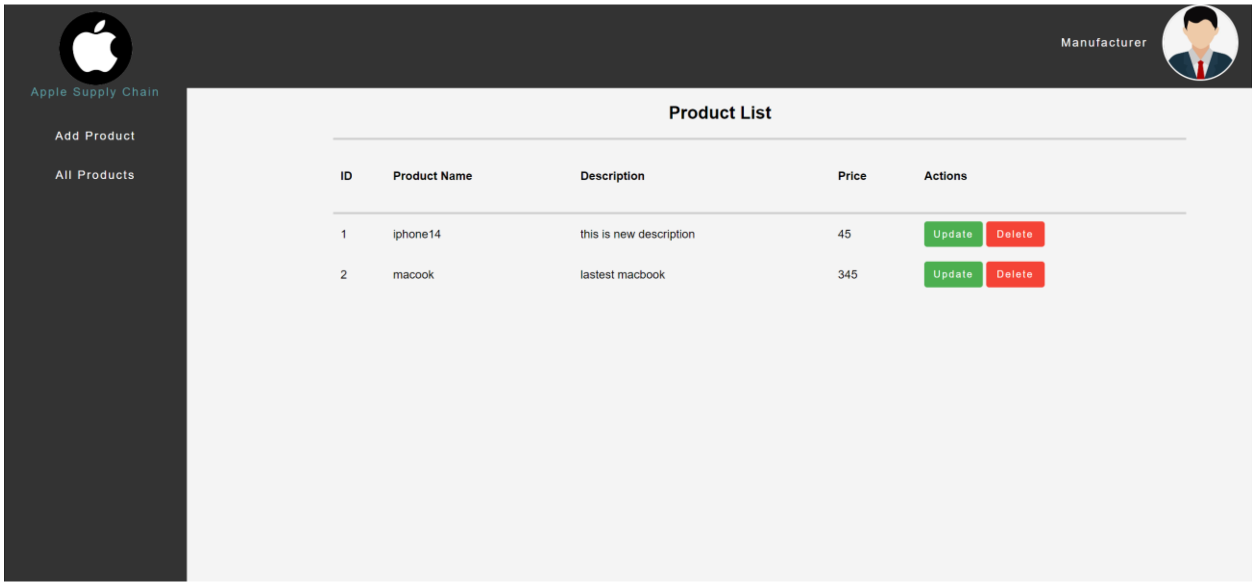


localhost:3000 says

Product uploaded successfully!

Product ID:	2
Name:	macook
Description:	lastest macbook
Price:	345

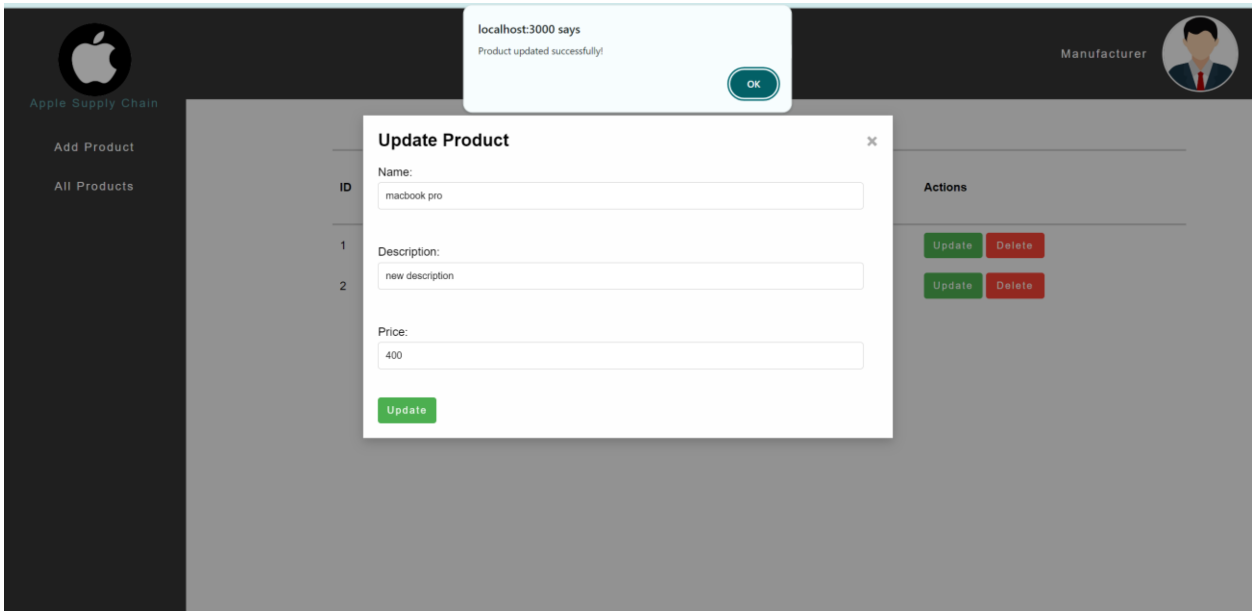
Upon addin the details thses details will be fetched and displayed in the all product :



Product List

ID	Product Name	Description	Price	Actions
1	iphone14	this is new description	45	Update Delete
2	macook	lastest macbook	345	Update Delete

The manufacturer can update or delete the product details form the ledger if the information they provided is incorrect or made a mistake while entering the data.
update:



localhost:3000 says
Product updated successfully!

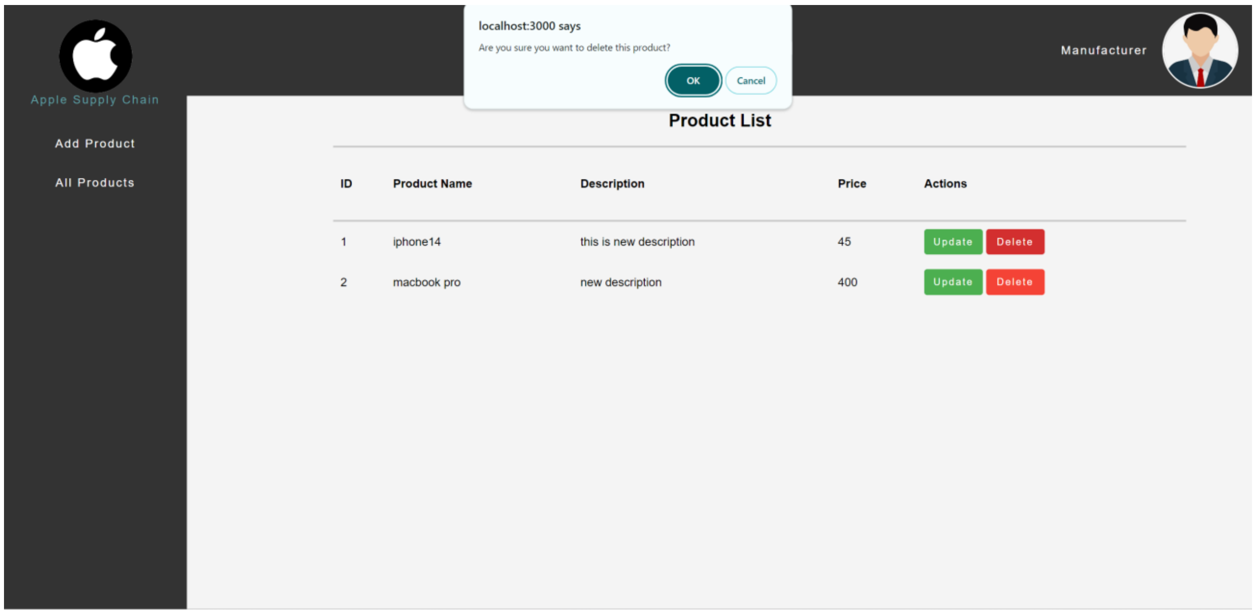
Update Product

Name: macbook pro

Description: new description

Price: 400

Delete:



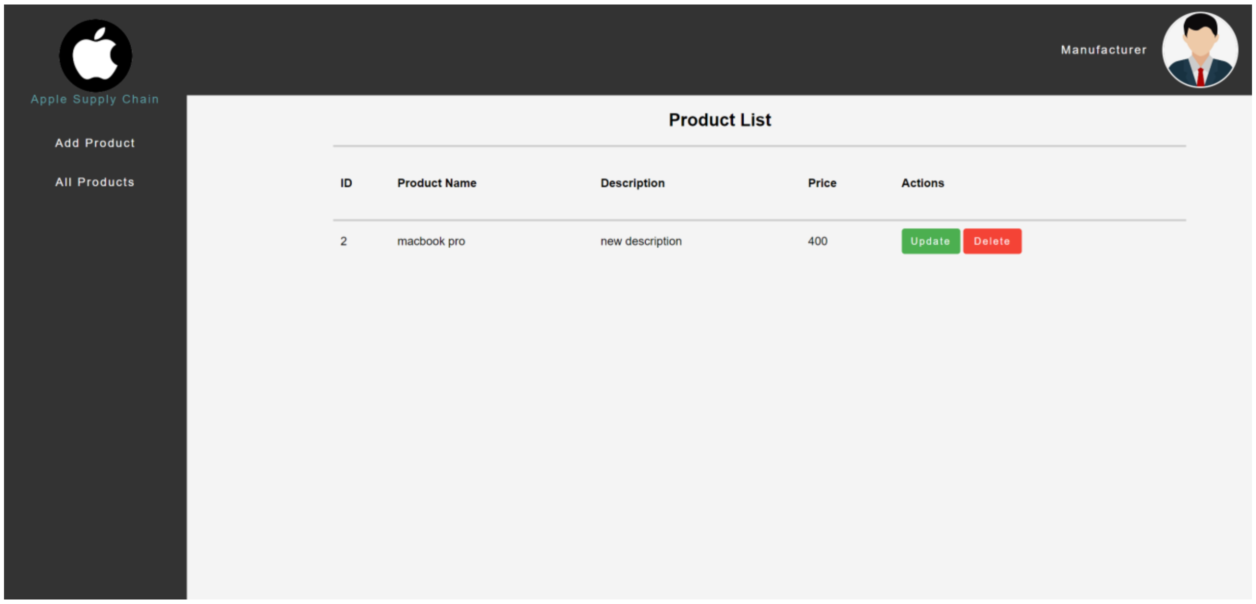
localhost:3000 says

Are you sure you want to delete this product?

Product List

ID	Product Name	Description	Price	Actions
1	iphone14	this is new description	45	Update Delete
2	macbook pro	new description	400	Update Delete

After deleting:



Product List

ID	Product Name	Description	Price	Actions
2	macbook pro	new description	400	Update Delete