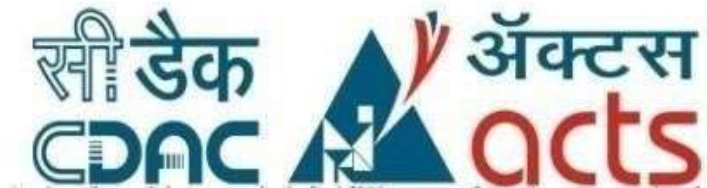


Project Report
On
Textile Supply Chain Management System using Hyperledger Fabrics



Submitted
In partial fulfilment
For the award of the Degree of
PG-Diploma in Fintech and Blockchain Development
(C-DAC, ACTS (Patna))

Guided By: Submitted By
Mr. Deavyansh Gautam

(1) Paras Manchanda(220980742009)
(2) Prashant Pandey(220980742011)
(3) Ram Gawas (220980742004)

Centre for Development of Advanced Computing (C-DAC), ACTS (Patna-800001)

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, Mr. Deavyansh Gautam, C-DAC ACTS, Patna for her constant guidance and helpful suggestion for preparing this project Textile Supply Chain Management System using Hyperledger Fabrics

. We express our deep gratitude towards him for his inspiration, personal involvement, and constructive criticism that he provided us along with technical guidance during the course of this project.

We take this opportunity to thank the Head of the department Mr. Saket Jha for providing us with such a great infrastructure and environment for our overall development.

We express sincere thanks to Shri Aditya Kumar Sinha, Director of C-DAC Patna, for their kind cooperation and support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude toward Mr. Shivansh Raghuvanshi (Course Coordinator, PG-DFBD) for their valuable guidance and constant support throughout this work and helps to pursue additional studies.

Also our warm thanks to C-DAC ACTS, Patna which provides us with this opportunity to carry out this prestigious Project and to enhance our learning in various technical fields.

ABSTRACT

The textile industry has a complex and global supply chain, which includes multiple stakeholders such as farmers, manufacturers, suppliers, and retailers. This complexity can lead to challenges in managing the supply chain, such as lack of transparency, inefficient processes, and difficulties in tracking the movement of raw materials and finished products.

Blockchain technology, specifically Hyperledger Fabric, can provide a solution to these challenges by enabling secure and transparent data sharing among the stakeholders in the supply chain. In this paper, we propose a Textile Supply Chain Management System using Hyperledger Fabric, which can help to streamline the supply chain, reduce costs, and improve traceability.

The proposed system will allow each participant in the supply chain to have their own node on the Hyperledger Fabric network, which will enable them to see relevant data related to their transactions while keeping other data confidential. The system will utilize features such as private channels, smart contracts, and consensus protocols to ensure secure and efficient data sharing.

With the proposed system, companies in the textile industry can track the provenance of raw materials, such as cotton or wool, from the farm to the factory, and ultimately to the finished product. This can help ensure ethical sourcing of materials and provide customers with transparent information about the products they are buying.

Overall, the proposed Textile Supply Chain Management System using Hyperledger Fabric has the potential to revolutionize the textile industry supply chain by providing greater transparency, efficiency, and traceability.

Table of Contents

S. No	Title	Page No.
	Front Page Acknowledgement Abstract	I II III
	Table of Contents	IV
1	Introduction Problem Statement	
2	Methodology/ Techniques	02-16
2.1	Introduction	02
2.2	Approach and Methodology/ Techniques	03
2.3	Dataset Preparation	03
2.4	Model Description	06
3	Implementation	17-20
3.1	Implementation	17
4	Results	21-26
4.1	Results	21
5	Conclusion	27
5.1	Conclusion	27
6	References	28
6.1	References	28

Introduction

The textile industry is a complex and global supply chain involving multiple stakeholders such as manufacturers, suppliers, and retailers. This complexity can lead to challenges in managing the supply chain, such as lack of transparency, inefficient processes, and difficulties in tracking the movement of raw materials and finished products.

Blockchain technology has the potential to provide a solution to these challenges by enabling secure and transparent data sharing among the stakeholders in the supply chain. Hyperledger Fabric, a permissioned blockchain framework, is particularly suited for supply chain management due to its privacy and permission features.

In this paper, we propose a Textile Supply Chain Management System using Hyperledger Fabric. The proposed system will enable companies in the textile industry to track the provenance of raw materials, such as cotton or wool, from the farm to the factory, and ultimately to the finished product. This will ensure the ethical sourcing of materials and provide customers with transparent information about the products they are buying.

The proposed system will utilize Hyperledger Fabric's features such as private channels, smart contracts, and consensus protocols to ensure secure and efficient data sharing among the stakeholders in the supply chain. Each participant in the supply chain will have their own node on the Hyperledger Fabric network, which will enable them to see relevant data related to their transactions while keeping other data confidential.

The rest of this paper is organized as follows. In the next section, we will provide an overview of Hyperledger Fabric and its features. Then, we will discuss the proposed Textile Supply Chain Management System in detail, including its architecture, data flow, and use cases. Finally, we will conclude the paper with a discussion of the potential benefits of the proposed system for the textile industry.

Methodology and Techniques

Introduction:

Methodology and Techniques:

The proposed Textile Supply Chain Management System using Hyperledger Fabric will utilize several key techniques and methodologies to ensure efficient and secure data sharing among the stakeholders in the supply chain. These include:

Hyperledger Fabric Framework: The proposed system will be built on the Hyperledger Fabric framework, which is a permissioned blockchain framework designed for enterprise use. Hyperledger Fabric provides features such as private channels, smart contracts, and consensus protocols that are essential for supply chain management.

Smart Contracts: Smart contracts will be used to automate and execute the business logic of the proposed system. This will ensure that transactions are executed correctly and transparently among the stakeholders in the supply chain.

Private Channels: Private channels will be used to enable secure and confidential data sharing among the stakeholders in the supply chain. Each participant in the supply chain will have access only to the relevant data related to their transactions.

Consensus Protocols: Consensus protocols will be used to ensure that all participants in the supply chain agree on the state of the ledger. This will ensure that transactions are executed correctly and transparently.

Integration with Existing Systems: The proposed system will be designed to integrate with existing systems used by the stakeholders in the supply chain. This will ensure that the proposed system can be easily adopted and integrated into the existing supply chain infrastructure.

User Interface: The proposed system will have a user interface that will enable the stakeholders in the supply chain to easily access and interact with the system. The user interface will be designed to be user-friendly and intuitive.

Overall, the proposed Textile Supply Chain Management System using Hyperledger Fabric will utilize several key techniques and methodologies to ensure efficient and secure data sharing among the stakeholders in the supply chain. These techniques and methodologies

will be essential for ensuring the success of the proposed system and its adoption by the textile industry.

Approach & Methodology/Techniques:

Overall, the proposed Textile Supply Chain Management System using Hyperledger Fabric will utilize several key techniques and methodologies to ensure efficient and secure data sharing among the stakeholders in the supply chain. These techniques and methodologies will be essential for ensuring the success of the proposed system and its adoption by the textile industry.

Problem Statement

Problem Statement: Inefficient supply chain management leading to high lead times, high costs, low transparency, and poor-quality control in the textile industry. The textile industry involves multiple stakeholders, including suppliers, manufacturers, distributors, and retailers, making the supply chain complex and fragmented. The supply chain management in the textile industry is often inefficient, leading to long lead times, high costs, low transparency, and poor-quality control.

2. Goals:

Create a ledger system that creates an eco-system where information flows openly in a permissioned manner, reduces the assumed risk in the supply chain, and reduces the total cost of the supply chain while making the supply chain more agile and adaptive.

3. Exceptions:

An audit may need to be done and access may be needed in the case of a government investigation (E.g., Money laundering, illegal smuggling)

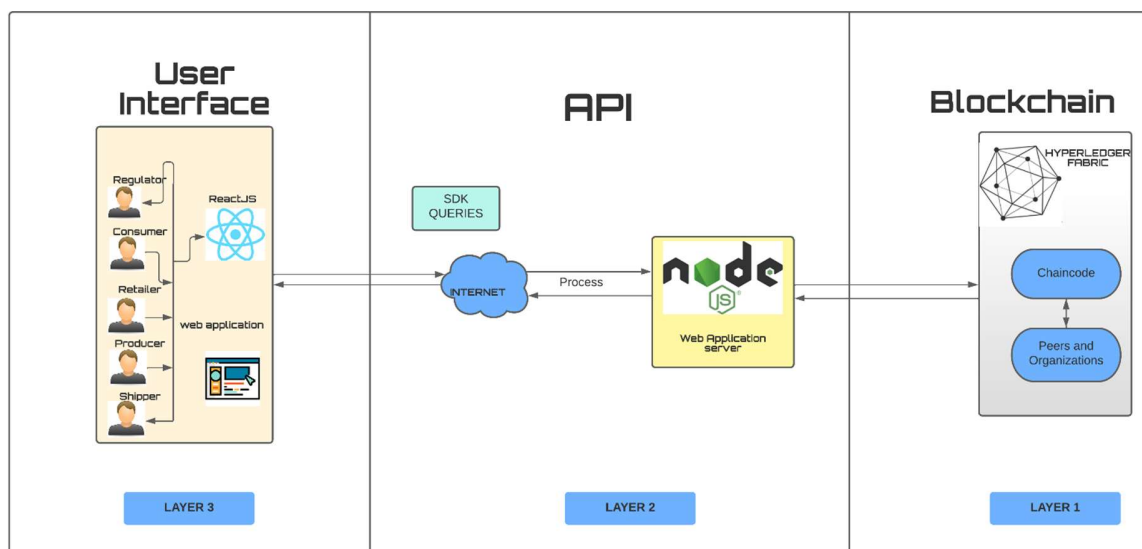
Architecture:

Hyperledger Fabric Blockchain framework,

Nodejs frontend and backend (blockchain) to communicate together,

Database CouchDB.

React is used for developing the Web Applications.



Project flow

Hyperledger Fabric is a permissioned blockchain framework designed to be modular and scalable for enterprise use cases. Its architecture is based on a distributed system of nodes that communicate and reach consensus on transactions through a consensus protocol.

Here is a brief overview of the architecture of Hyperledger Fabric:

Peer nodes: These are nodes that validate and endorse transactions. They maintain copies of the ledger and execute chaincode (smart contracts) to enforce business logic.

Orderer nodes: These nodes are responsible for ensuring the ordering and delivery of transactions to the peer nodes. They create blocks of transactions that are distributed to the peer nodes for validation and consensus.

Membership service provider (MSP): This is a component that manages identities and permissions of network participants. It provides authentication and authorization services for the nodes and users in the network.

Chaincode: Chaincode is the smart contract logic that defines the business rules and workflows of the blockchain application. It is executed by the peer nodes and can interact with external data sources and other chaincodes.

5.Channels: Channels provide a way to create separate sub-networks within the overall blockchain network, where only certain participants have access. This enables different organizations to transact privately and securely without the need for a separate blockchain network.

6.Ledger: The ledger is a distributed database that records all transactions and state changes on the blockchain. It is maintained by the peer nodes and can be configured for different types of data storage and access control.

Node.js frontend and backend (blockchain) to communicate together,

In order to enable communication between a Node.js frontend and backend for a blockchain application, there are a few steps that need to be taken.

Setting up a Node.js backend: The first step is to set up a Node.js backend that will communicate with the blockchain. This can be done using a blockchain SDK or API, such as the Hyperledger Fabric Node.js SDK.

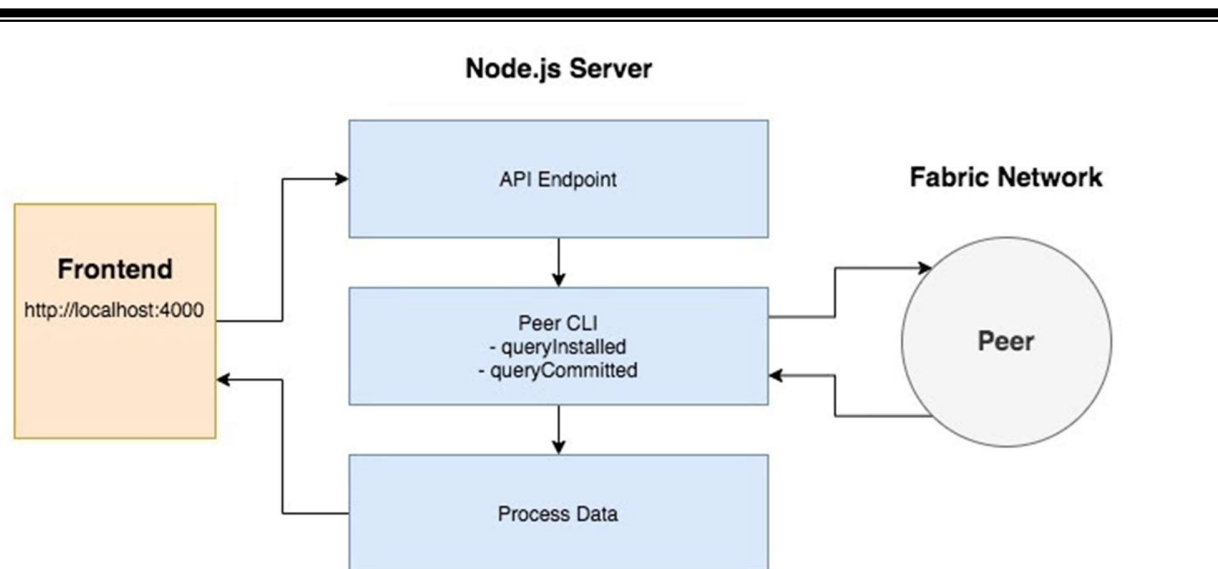
Creating an API: Once the backend is set up, you need to create an API that will expose the blockchain functionality to the frontend. This can be done using an API framework such as Express.js or Hapi.js.

Connecting the frontend to the backend: To enable communication between the frontend and backend, you can use an HTTP client library such as Axios or the built-in fetch API. This allows the frontend to make requests to the backend API and receive responses.

Handling responses: Once the frontend sends a request to the backend, it will receive a response. The frontend can then process this response and update the UI accordingly.

Authenticating users: To ensure that only authorized users can access the blockchain functionality, you can implement authentication and authorization mechanisms on both the frontend and backend.

Overall, the key to enabling communication between a Node.js frontend and backend for a blockchain application is to use a well-defined API and HTTP client library to send requests and receive responses.



COUCHDB

CouchDB is a NoSQL database management system that uses a document-oriented model to store data. It is open-source, highly scalable, and designed for distributed environments.

Some key features of CouchDB include:

Schema-less data model: CouchDB is a document-oriented database, which means that it does not require a predefined schema. Instead, data is stored as JSON documents, which are flexible and can be easily updated or modified.

Replication and synchronization: CouchDB allows for easy replication and synchronization of data across multiple nodes, which makes it highly scalable and fault-tolerant.

MapReduce: CouchDB uses MapReduce for querying and data aggregation, which allows for efficient processing of large amounts of data.

RESTful API: CouchDB has a RESTful API that allows for easy integration with web applications and other databases.

Security: CouchDB supports authentication and authorization mechanisms to ensure data security and privacy.

Overall, CouchDB is a highly versatile and scalable NoSQL database management system that is well-suited for distributed environments and applications that require flexible data models. It can be used in a variety of industries and use cases, such as healthcare, finance, and e-commerce.

React is used for developing the Web Applications.

React is a popular JavaScript library used for building user interfaces and web applications. It was developed by Facebook and has become widely adopted by developers due to its simplicity, flexibility, and performance.

Some of the key features and benefits of React for developing web applications include:

Component-based architecture: React uses a component-based architecture, which makes it easy to reuse code and build complex user interfaces. Components are reusable and can be easily combined to create larger, more complex components.

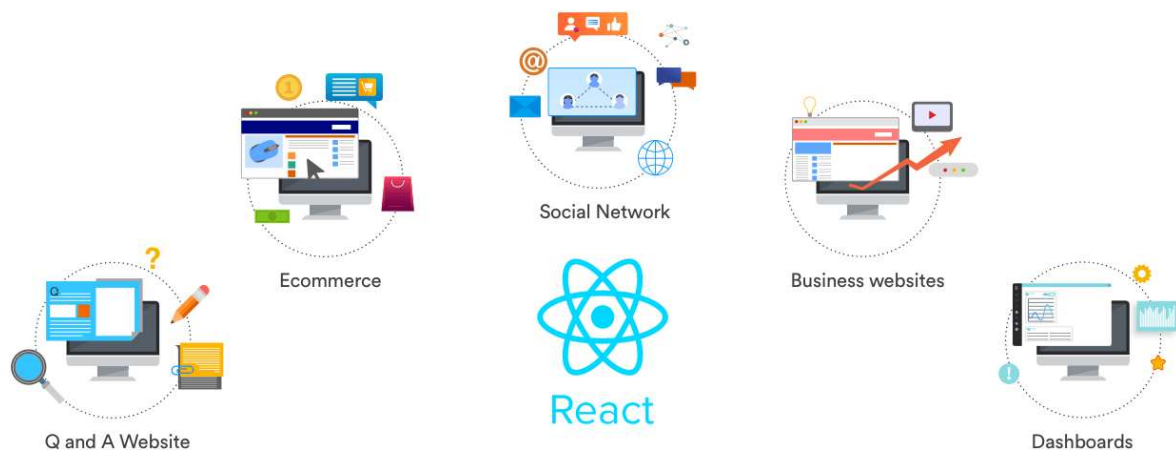
Virtual DOM: React uses a virtual DOM (Document Object Model) to efficiently update the UI. The virtual DOM is a lightweight representation of the actual DOM and allows React to update the UI without reloading the entire page.

Performance: React is designed for high performance and can handle large and complex user interfaces without sacrificing performance.

JSX: React uses JSX, a syntax extension that allows developers to write HTML-like code within JavaScript. This makes it easy to create and manipulate UI components.

Large community: React has a large and active community of developers, which means that there are many resources available for learning and problem-solving.

Overall, React is a powerful and flexible tool for building web applications. Its component-based architecture, virtual DOM, and performance make it a popular choice for developers who want to create complex user interfaces quickly and efficiently.



User Stories:

This is a great example of how Hyperledger Fabric can be used to build an end-to-end blockchain application with attribute-based access control, user management, and a React front-end UI. The application provides end-to-end visibility of the supply chain, allowing customers to trace the order history from ordering to shipping and receiving.

Some key features of this application include:

Attribute-based access control: The application implements attribute-based access control to ensure that only authorized users can access and modify data on the blockchain. This provides an additional layer of security and privacy for sensitive data.

User management: The application includes a user management system that allows administrators to create and manage user accounts, set permissions, and enforce access control policies.

React front-end UI: The application uses React to create a user-friendly front-end UI that allows users to interact with the blockchain ledger, query data, and view order history.

Supply chain visibility: The application provides end-to-end visibility of the supply chain, allowing customers to trace the order history from ordering to shipping and receiving. This provides transparency and accountability throughout the supply chain.

Overall, this example demonstrates how Hyperledger Fabric can be used to build a secure, scalable, and transparent blockchain application that provides end-to-end visibility of the

supply chain. It also showcases the benefits of using React for developing the front-end UI, which allows for a user-friendly and responsive interface.

STAKE HOLDERS

The stakeholders in this blockchain application are:

Retailer: They place an order with the producer and receive the order from the shipper. They have a vested interest in ensuring that the products they receive are of high quality and are delivered on time.

Producer: They fulfill orders and assign them to shippers. They have a vested interest in ensuring that they produce high-quality products and deliver them on time.

Shipper: They create shipments and transport orders assigned by the producer. They have a vested interest in ensuring that the products are transported safely and delivered on time.

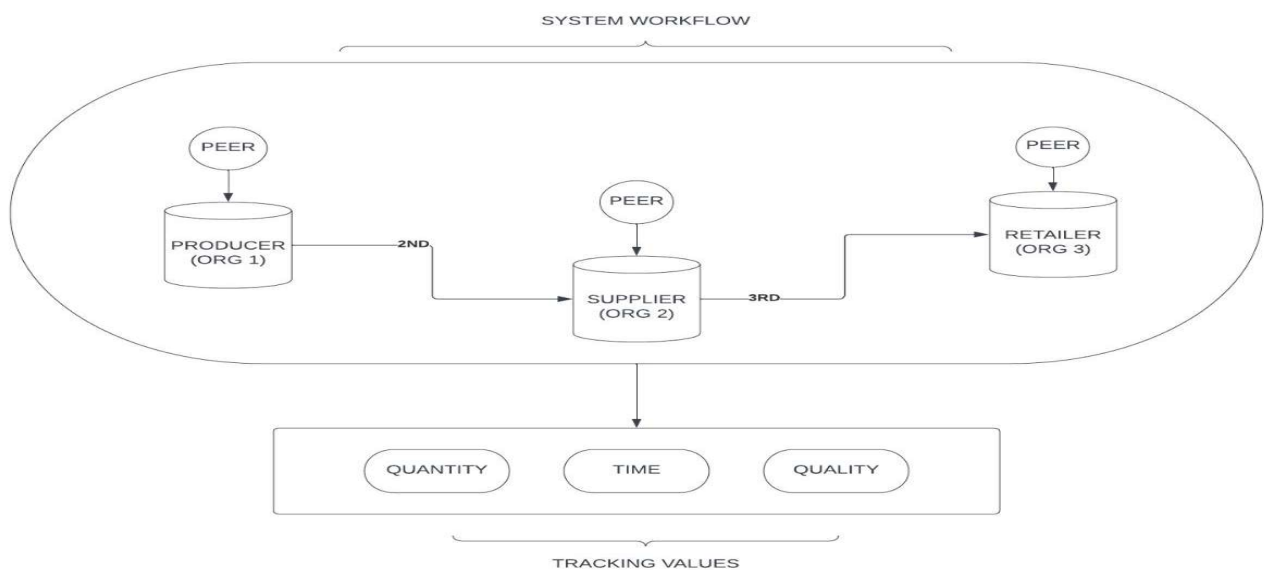
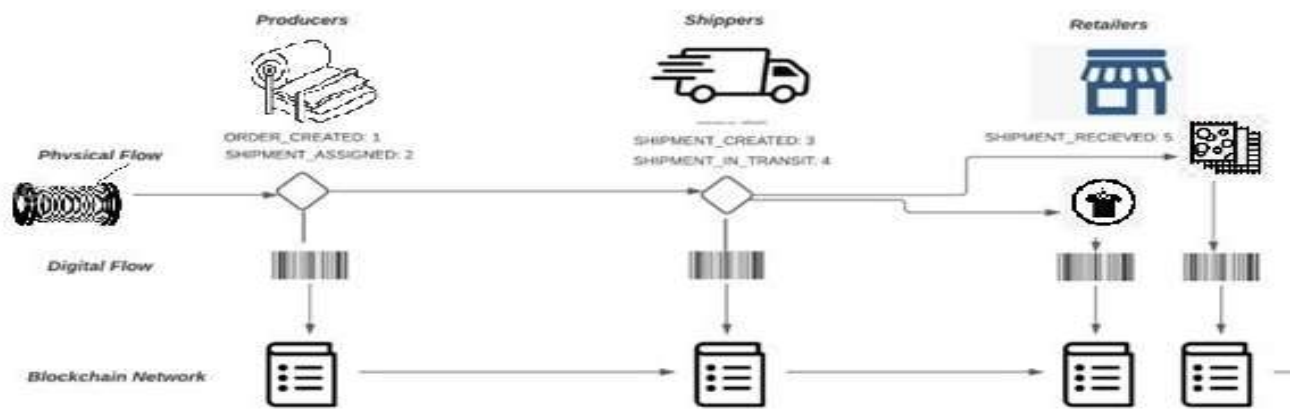
Customer: They query an order to get the order transaction history, tracing it back to its origination. They have a vested interest in ensuring that the products they receive are of high quality and have been transported safely.

Regulator: They moderate all orders in the system to ensure that proper quality and guidelines are being followed for audit purposes. They have a vested interest in ensuring that the products being produced and transported meet regulatory standards and guidelines.

Overall, these stakeholders have different roles and interests in the supply chain, but all of them have a common interest in ensuring that the products being produced and transported are of high quality and meet regulatory standards. The blockchain application provides transparency and accountability throughout the supply chain, which benefits all stakeholders involved.

Roles	Access Type
Admin	Create users
Regulator	Access order history for audit purpose
Producer	Create order, assign shipper
Retailer	Receive shipment
Shipper	Ship order to retailer
Customer	View order history of items

2. State diagram



A textile supply chain management system typically involves the following flow:

In a textile supply chain management system, each entity can track the movement of the fabric or finished products from one entity to another using a blockchain-based platform. This platform can capture and store data about the product, including its origin, quality, and journey through the supply chain. This enables transparency, traceability, and accountability in the supply chain, and helps to prevent issues such as fraud, and unethical practices.

Producer (Manufacturers): This is the entity that creates the raw material (such as cotton) or manufactures the fabric. The producer is responsible for sourcing raw materials, processing and manufacturing fabrics, and ensuring quality control.

The fabric is further processed by manufacturers who convert it into finished products such as clothing, home textiles, or industrial textiles. Manufacturers are responsible for ensuring quality, managing production schedules, and delivering finished products to retailers or distributors.

Shippers: The fabric produced by the producer is shipped to the next entity in the supply chain. Shippers are responsible for logistics, warehousing, and transportation of the fabric from the producer to the next entity.

Retailers: The finished products are sold by retailers to consumers. Retailers are responsible for stocking and displaying products, managing inventory, and providing customer service.

Consumers: The end-users who purchase and use the finished products.

Through this platform, producers, shippers, retailers, and consumers can all access real-time data about the product, including its location, quality, and status, and take informed decisions based on this information. This improves efficiency, reduces costs, and enhances customer trust and satisfaction.

IMPLEMENTATION

NETWORK SETUP

Open a terminal window (Command Prompt or PowerShell).

Run “docker –version” to ensure that you have a supported version of Docker:

Pull the [hello-world image](#) from Docker Hub and run a container:

```
$ docker run hello-world
```

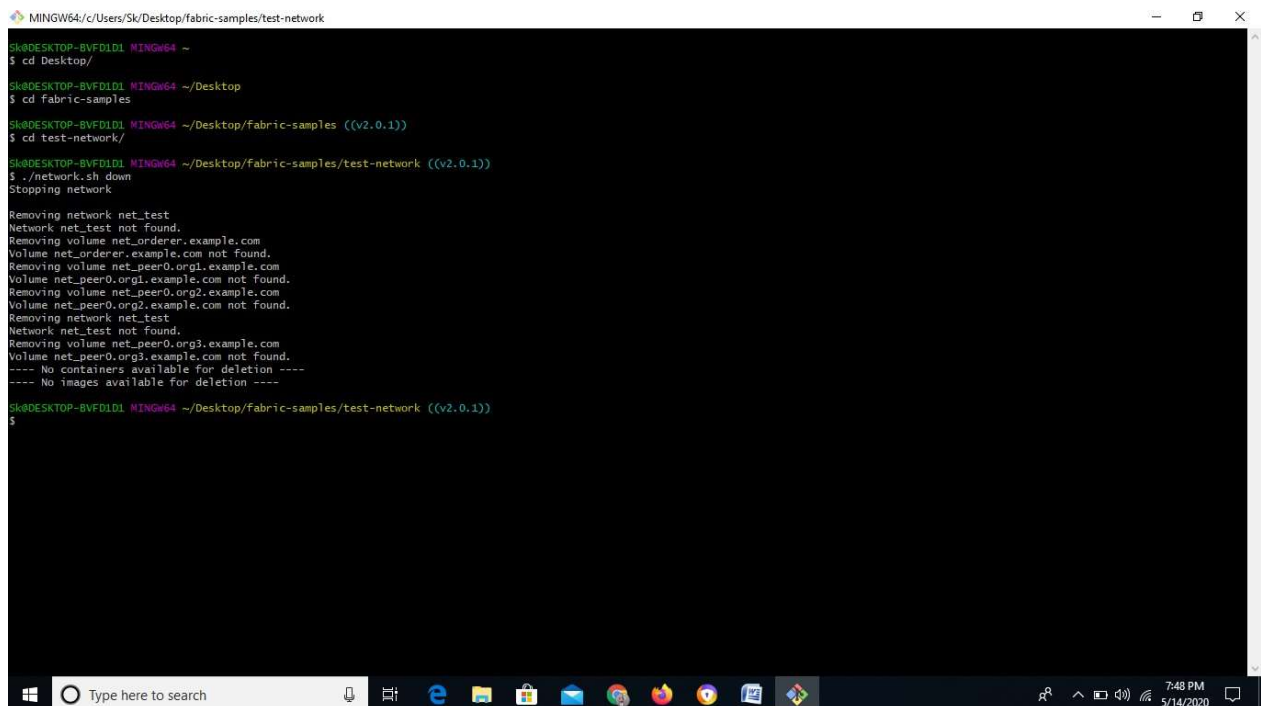
```
$ docker container ls --all
```

Install node.js

Setting Up the Hyperledger Sample in Local machine

```
curl -sSL https://bit.ly/2ysbOFE | bash -s
```

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



```
MINGW64/c/Users/Sk/Desktop/fabric-samples/test-network
Sk@DESKTOP-BVFD1D1 MINGW64 ~
$ cd Desktop/
Sk@DESKTOP-BVFD1D1 MINGW64 ~/Desktop
$ cd fabric-samples
Sk@DESKTOP-BVFD1D1 MINGW64 ~/Desktop/fabric-samples ((v2.0.1))
$ cd test-network/
Sk@DESKTOP-BVFD1D1 MINGW64 ~/Desktop/fabric-samples/test-network ((v2.0.1))
$ ./network.sh down
Stopping network
Removing network net_test
Network net_test not found.
Removing volume net_orderer.example.com
Volume net_orderer.example.com not found.
Removing volume net_peer0.org1.example.com
Volume net_peer0.org1.example.com not found.
Removing volume net_peer0.org2.example.com
Volume net_peer0.org2.example.com not found.
Removing volume net_peer0.org3.example.com
Volume net_peer0.org3.example.com not found.
---- No containers available for deletion ----
---- No images available for deletion ----
Sk@DESKTOP-BVFD1D1 MINGW64 ~/Desktop/fabric-samples/test-network ((v2.0.1))
$
```

./network.sh up

```
MINGW64/c/Users/Sk/Desktop/fabric-samples/test-network
SK@DESKTOP-BVFD1D1 MINGW64 ~/Desktop/fabric-samples/test-network ((v2.0.1))
$ ./network.sh up
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb' with crypto from 'cryptogen'

LOCAL_VERSION=2.1.0
DOCKER_IMAGE_VERSION=2.1.0
/c/Users/Sk/Desktop/fabric-samples/bin/cryptogen

##### Generate certificates using cryptogen tool #####
##### Create Org1 Identities #####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organizations
org1.example.com
+ res=0
+ set +x
##### Create Org2 Identities #####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organizations
org2.example.com
+ res=0
+ set +x
##### Create Orderer Org Identities #####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organizations
+ res=0
+ set +x

Generate CCP files for Org1 and Org2
/c/Users/Sk/Desktop/fabric-samples/bin/configtxgen
##### Generating Orderer Genesis block #####
+ configtxgen -profile TwoOrgsOrdererGenesis -channelID system-channel -outputBlock ./system-genesis-block/genesis.block
[2020-05-14 18:44:14.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 Loading configuration
[2020-05-14 18:44:15.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 Orderer.Addresses unset, setting to [127.0.0.1:7050]
[2020-05-14 18:44:15.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 orderer type: etcdraft
[2020-05-14 18:44:15.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" election_tick:10 heartbea
t_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
[2020-05-14 18:44:15.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 Loaded configuration: C:\Users\Sk\Desktop\fabric-samples\test-network\configtx\configtx.yaml
[2020-05-14 18:44:15.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 Generating genesis block
[2020-05-14 18:44:15.888] [207] [common.tools.configtxgen.localconfig] main => [200] 000 Writing genesis block
+ res=0
+ set +x
Creating network "net_test" with the default driver
Creating volume "net_orderer.example.com" with default driver
```

```
MINGW64/c/Users/Sk/Desktop/fabric-samples/test-network
Sk@DESKTOP-EVFD1D1 MINGW64 ~/Desktop/fabric-samples/test-network ((v2.0.1))
$ ./network.sh createChannel
Creating channel 'mychannel'.

If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb'

### Generating channel configuration transaction 'mychannel.tx' ###
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/mychannel.tx -channelID mychannel
2020-05-14 19:10:10.874 [INF] [common.configtxgen] > args => [200] 000 Loading configuration
2020-05-14 19:10:10.780 [INF] [common.configtxgen] > loadConfig => [200] 000 Loaded configuration: C:\Users\Sk\Desktop\fabric-samples\test-network\configtx\configtx.yaml
2020-05-14 19:10:10.780 [INF] [common.configtxgen] > defaultOutputChannelMetadata => [200] 000 Generating new channel configtx
2020-05-14 19:10:10.718 [INF] [common.configtxgen] > defaultOutputChannelMetadata => [200] 000 Writing new channel tx
+ res=0
+ set +x

### Generating channel configuration transaction 'mychannel.tx' ###
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP
2020-05-14 19:10:10.861 [INF] [common.configtxgen] > args => [200] 000 Loading configuration
2020-05-14 19:10:10.780 [INF] [common.configtxgen] > loadConfig => [200] 000 Loaded configuration: C:\Users\Sk\Desktop\fabric-samples\test-network\configtx\configtx.yaml
2020-05-14 19:10:10.868 [INF] [common.configtxgen] > defaultOutputAnchorPeersUpdate => [200] 000 Generating anchor peer update
2020-05-14 19:10:10.897 [INF] [common.configtxgen] > defaultOutputAnchorPeersUpdate => [200] 000 Writing anchor peer update
+ res=0
+ set +x

##### Generating anchor peer update for Org2MSP #####
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP
2020-05-14 19:10:10.862 [INF] [common.configtxgen] > args => [200] 000 Loading configuration
2020-05-14 19:10:10.780 [INF] [common.configtxgen] > loadConfig => [200] 000 Loaded configuration: C:\Users\Sk\Desktop\fabric-samples\test-network\configtx\configtx.yaml
2020-05-14 19:10:10.862 [INF] [common.configtxgen] > defaultOutputAnchorPeersUpdate => [200] 000 Generating anchor peer update
2020-05-14 19:10:10.873 [INF] [common.configtxgen] > defaultOutputAnchorPeersUpdate => [200] 000 Writing anchor peer update
+ res=0
+ set +x

Creating channel mychannel
Using organization 1
+ peer channel create -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.example.com -f ./channel-artifacts/mychannel.tx --outputBlock ./channel-artifacts/mychannel.block --tl
+ res=0
+ set +x

2020-05-14 19:10:10.780 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.865 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[NOT_FOUND]
2020-05-14 19:10:10.865 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.112 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[SERVICE_UNAVAILABLE]
2020-05-14 19:10:10.120 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.120 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[SERVICE_UNAVAILABLE]
2020-05-14 19:10:10.121 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.121 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[SERVICE_UNAVAILABLE]
2020-05-14 19:10:10.121 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.121 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[SERVICE_UNAVAILABLE]
2020-05-14 19:10:10.121 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
```

```
MINGW64/c/Users/Sk/Desktop/fabric-samples/test-network
2020-05-14 19:10:10.145 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[SERVICE_UNAVAILABLE]
2020-05-14 19:10:10.145 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.145 [INF] [cli.common] > readBlock => [200] 000 Expect block, but got status: &[SERVICE_UNAVAILABLE]
2020-05-14 19:10:10.145 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.145 [INF] [cli.common] > readBlock => [200] 000 Received block: 0

===== Channel 'mychannel' created =====

Join Org1 peers to the channel...
Using organization 1
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
+ set +x
2020-05-14 19:10:10.868 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.121 [INF] [channelCmd] > executeJoin => [200] 000 Successfully submitted proposal to join channel

Join Org2 peers to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
+ set +x
2020-05-14 19:10:10.862 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.121 [INF] [channelCmd] > executeJoin => [200] 000 Successfully submitted proposal to join channel

Updating anchor peers for org1...
Using organization 1
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile ./channel-artifacts/organizations/example.com/orderers/orderer.example.com/msp/tlsca/tlsca.example.com-cert.pem
+ res=0
+ set +x
2020-05-14 19:10:10.862 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.864 [INF] [channelCmd] > update => [200] 000 Successfully submitted channel update
===== Anchor peers updated for org 'Org1MSP' on channel 'mychannel' =====

Updating anchor peers for org2...
Using organization 2
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c mychannel -f ./channel-artifacts/Org2MSPanchors.tx --tls true --cafile ./channel-artifacts/organizations/example.com/orderers/orderer.example.com/msp/tlsca/tlsca.example.com-cert.pem
+ res=0
+ set +x
2020-05-14 19:10:10.862 [INF] [channelCmd] > InitCmdFactory => [200] 000 Endorser and orderer connections initialized
2020-05-14 19:10:10.862 [INF] [channelCmd] > update => [200] 000 Successfully submitted channel update
===== Anchor peers updated for org 'Org2MSP' on channel 'mychannel' =====

----- Channel successfully joined -----

Sk@DESKTOP-EVFD1D1 MINGW64 ~/Desktop/fabric-samples/test-network ((v2.0.1))
$
```

If the command was successful, you can see the following message printed in C-DAC ACTS (Patna)/PG-DFBD

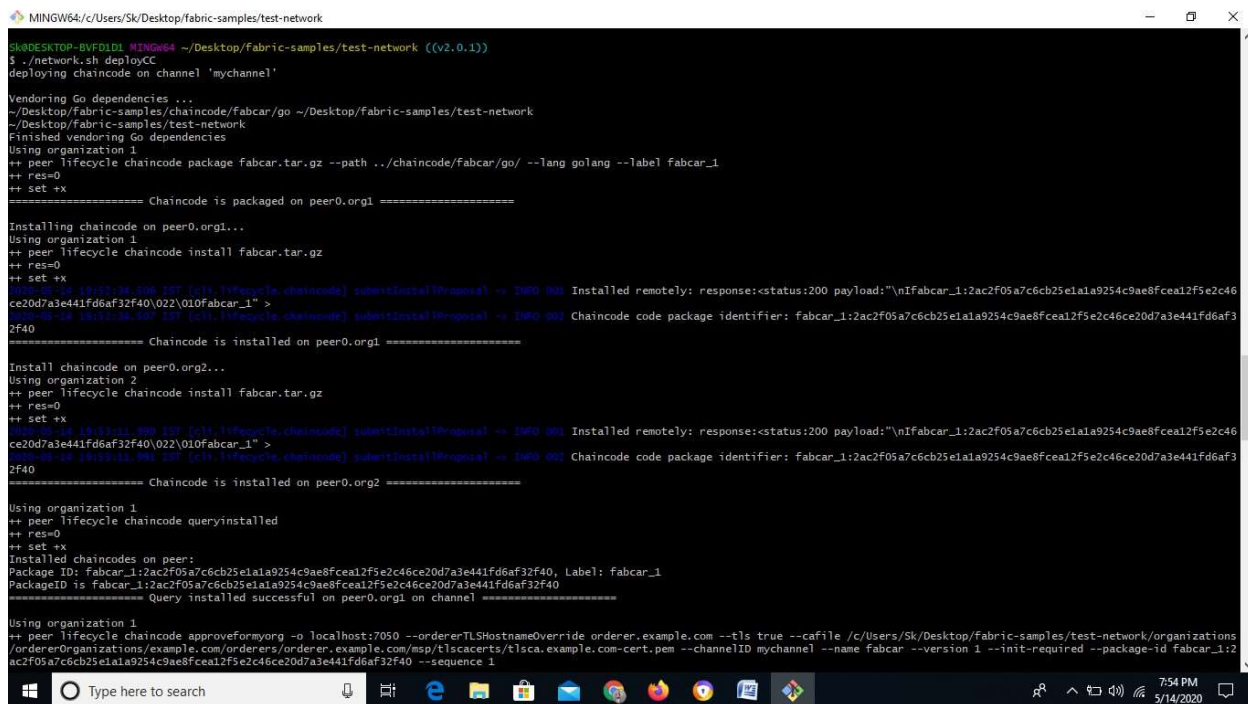
your logs:

===== Channel successfully joined =====

Starting a chaincode on the channel

After you have used the network.sh to create a channel, you can start a chaincode on the channel using the following command:

`./network.sh deployC`



```
MINGW64/c/Users/Sk/Desktop/fabric-samples/test-network ((v2.0.1))
$ ./network.sh deployC
deploying chaincode on channel 'mychannel'

Vendoring Go dependencies ...
~/Desktop/fabric-samples/chaincode/fabcar/go ~/Desktop/fabric-samples/test-network
~/Desktop/fabric-samples/test-network
Finished vendoring Go dependencies
Using organization 1
++ peer lifecycle chaincode package fabcar.tar.gz --path ../chaincode/fabcar/go/ --lang golang --label fabcar_1
++ res=0
++ set +x
===== Chaincode is packaged on peer0.org1 =====

Installing chaincode on peer0.org1...
Using organization 1
++ peer lifecycle chaincode install fabcar.tar.gz
++ res=0
++ set +x
[INFO] 04:19:23.04: 999 [20] [cli.lifecycle.chaincode] submitInstallProposal => [INFO] 999 Installed remotely: response:<status:200 payload:"\nIfabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40\022\010fabcar_1" >
[INFO] 04:19:23.04: 999 [20] [cli.lifecycle.chaincode] submitInstallProposal => [INFO] 999 Chaincode code package identifier: fabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40
===== Chaincode is installed on peer0.org1 =====

Install chaincode on peer0.org2...
Using organization 2
++ peer lifecycle chaincode install fabcar.tar.gz
++ res=0
++ set +x
[INFO] 04:19:23.04: 999 [20] [cli.lifecycle.chaincode] submitInstallProposal => [INFO] 999 Installed remotely: response:<status:200 payload:"\nIfabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40\022\010fabcar_1" >
[INFO] 04:19:23.04: 999 [20] [cli.lifecycle.chaincode] submitInstallProposal => [INFO] 999 Chaincode code package identifier: fabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40
===== Chaincode is installed on peer0.org2 =====

Using organization 1
++ peer lifecycle chaincode queryinstalled
++ res=0
++ set +x
Installed chaincodes on peer:
Package ID: fabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40, Label: fabcar_1
PackageID is fabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40
===== Query installed successful on peer0.org1 on channel =====

Using organization 1
++ peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls true --cafile ./Users/Sk/Desktop/fabric-samples/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem --channelID mychannel --name fabcar --version 1 --init-required --package-id fabcar_1:2ac2f05a7c6cb25e1a1a9254c9ae8fcea12f5e2c46ce20d7a3e441fd6af32f40 --sequence 1
```

```
MINGW64/c:/Users/Sk/Desktop/fabric-samples/test-network
Attempting to Query committed status on peer0.org1 on channel 'mychannel'...
++ peer lifecycle chaincode querycommitted --channelID mychannel --name fabcar
++ res=0
++ set +x

Committed chaincode definition for chaincode 'fabcar' on channel 'mychannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscv, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'mychannel'

Using organization 2
Attempting to Query committed status on peer0.org2 on channel 'mychannel'...
++ peer lifecycle chaincode querycommitted --channelID mychannel --name fabcar
++ res=0
++ set +x

Committed chaincode definition for chaincode 'fabcar' on channel 'mychannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscv, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'mychannel'

Using organization 1
++ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls true --cafile /c:/Users/Sk/Desktop/fabric-samples/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n fabcar --peerAddresses localhost:7051 --tlsRootCertFiles /c:/Users/Sk/Desktop/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles /c:/Users/Sk/Desktop/fabric-samples/test-network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt --isInit -c '{"function": "initLedger", "Args": []}'
++ res=0
++ set +x
===== Invoke transaction successful on peer0.org1 peer0.org2 on channel 'mychannel' =====
Chaincode invoke successful. result: status:200

Querying chaincode on peer0.org1...
Using organization 1
Attempting to Query peer0.org1 ...1589466245 secs
++ peer chaincode query -C mychannel -n fabcar -c '{"Args":["queryAllCars"]}'
++ res=0
++ set +x

[{"Key": "CAR0", "Record": {"make": "Toyota", "model": "Prius", "colour": "blue", "owner": "Tomoko"}}, {"Key": "CAR1", "Record": {"make": "Ford", "model": "Mustang", "colour": "red", "owner": "Brad"}}, {"Key": "CAR2", "Record": {"make": "Hyundai", "model": "Tucson", "colour": "green", "owner": "Jin Soo"}}, {"Key": "CAR3", "Record": {"make": "Volkswagen", "model": "Passat", "colour": "yellow", "owner": "Max"}}, {"Key": "CAR4", "Record": {"make": "Tesla", "model": "S", "colour": "black", "owner": "Adriana"}}, {"Key": "CAR5", "Record": {"make": "Peugeot", "model": "205", "colour": "purple", "owner": "Michel"}}, {"Key": "CAR6", "Record": {"make": "Chery", "model": "S22L", "colour": "white", "owner": "Aarav"}}, {"Key": "CAR7", "Record": {"make": "Fiat", "model": "Punto", "colour": "violet", "owner": "Pari"}}, {"Key": "CAR8", "Record": {"make": "Tata", "model": "Nano", "colour": "indigo", "owner": "Valeria"}}, {"Key": "CAR9", "Record": {"make": "Holden", "model": "Barina", "colour": "brown", "owner": "Shotaro"}}]
Query successful on peer0.org1 on channel 'mychannel'

Sk@DESKTOP-BVFD1DL MINGW64 ~/Desktop/fabric-samples/test-network ((v2.0.1))
$ |
```

CHAIN CODE DEPLOYMENT

Chain code deployment refers to the process of installing and instantiating smart contracts (chain code) onto a blockchain network. Chain code is the code that defines the rules and logic governing a blockchain network, and it is executed by the nodes on the network.

To deploy chain code onto a blockchain network, you typically need to follow these steps:

Write the chain code: This involves creating the code that will govern the behavior of the network.

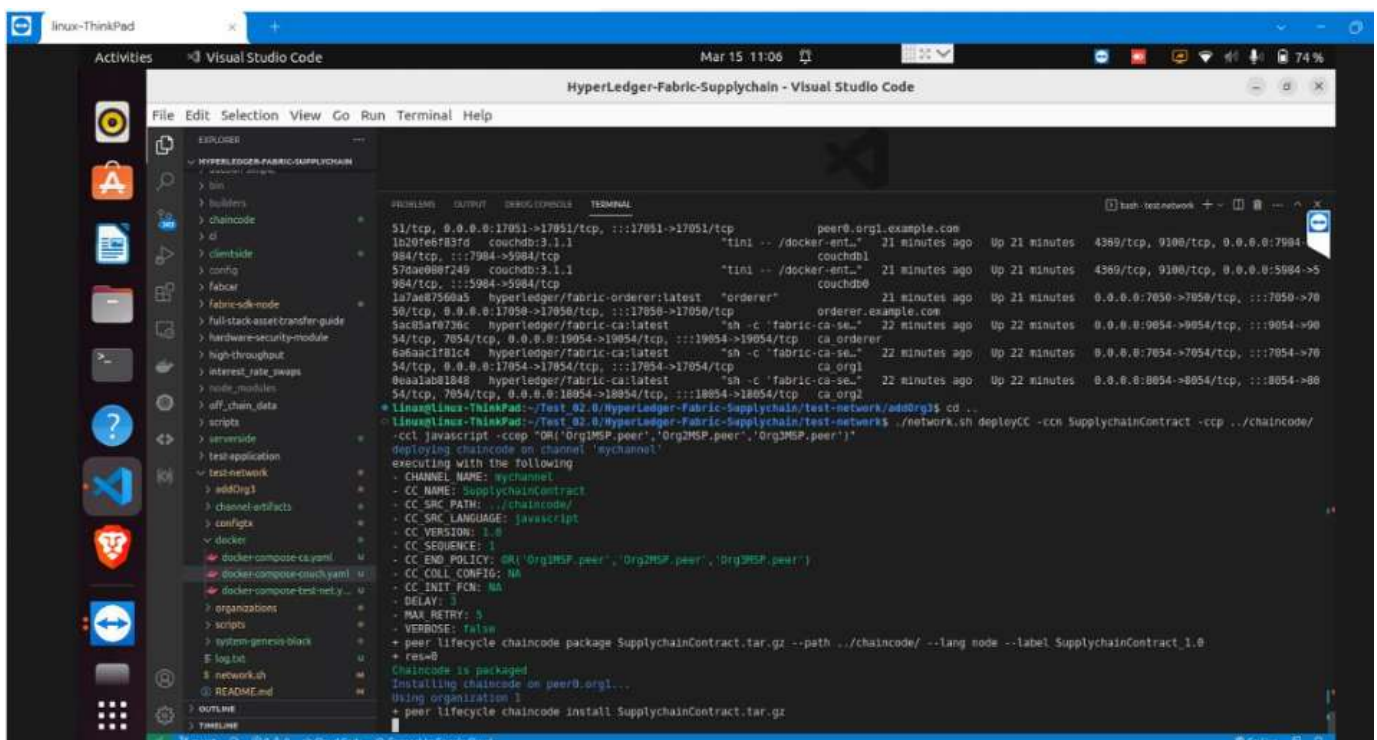
Package the chain code: The chain code needs to be packaged into a container or package format that can be easily installed onto the blockchain network.

Install the chain code: The packaged chain code needs to be installed onto the nodes in the blockchain network. This typically involves using a command-line tool or a web interface to upload the chain code.

Instantiate the chain code: Once the chain code is installed on the network, it needs to be instantiated, which means creating an instance of the code that can be used by network participants to interact with the network.

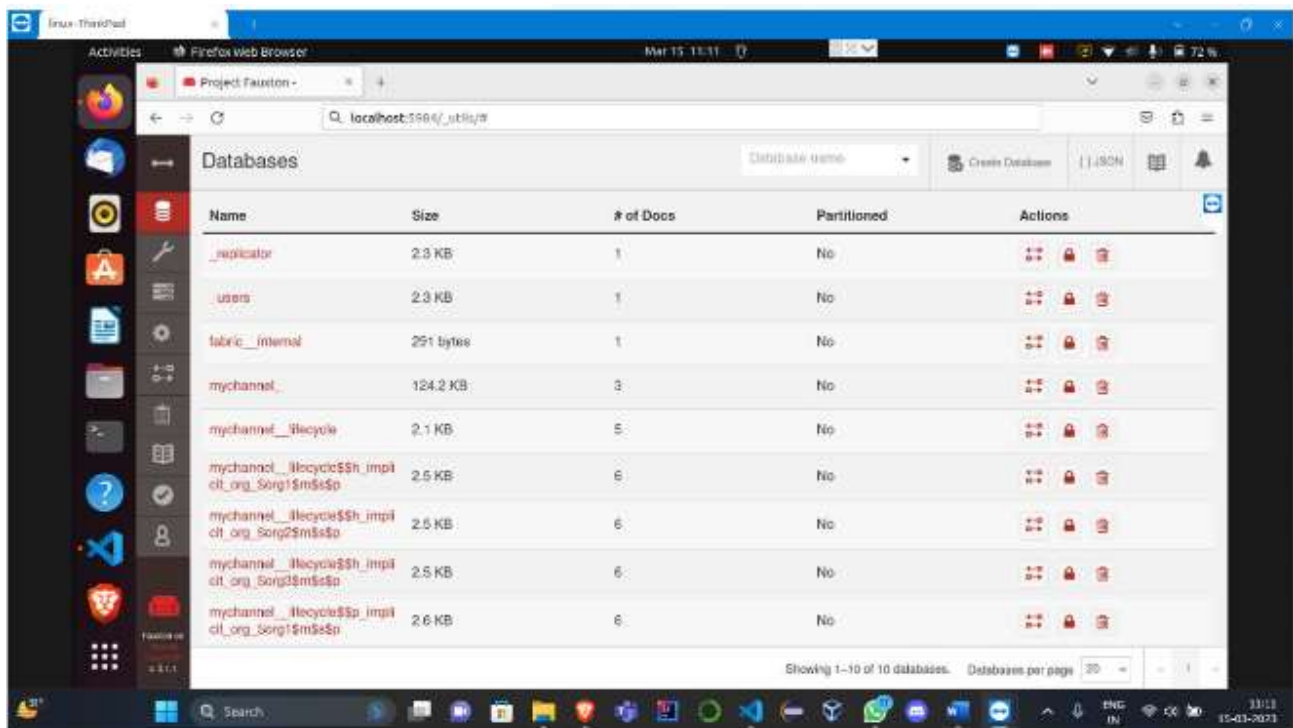
Test and debug the chain code: After the chain code is deployed, it needs to be thoroughly tested and debugged to ensure that it works as expected and is free from errors or vulnerabilities.

The exact process of deploying chain code can vary depending on the blockchain platform and the tools used, but these general steps provide a good overview of the process.

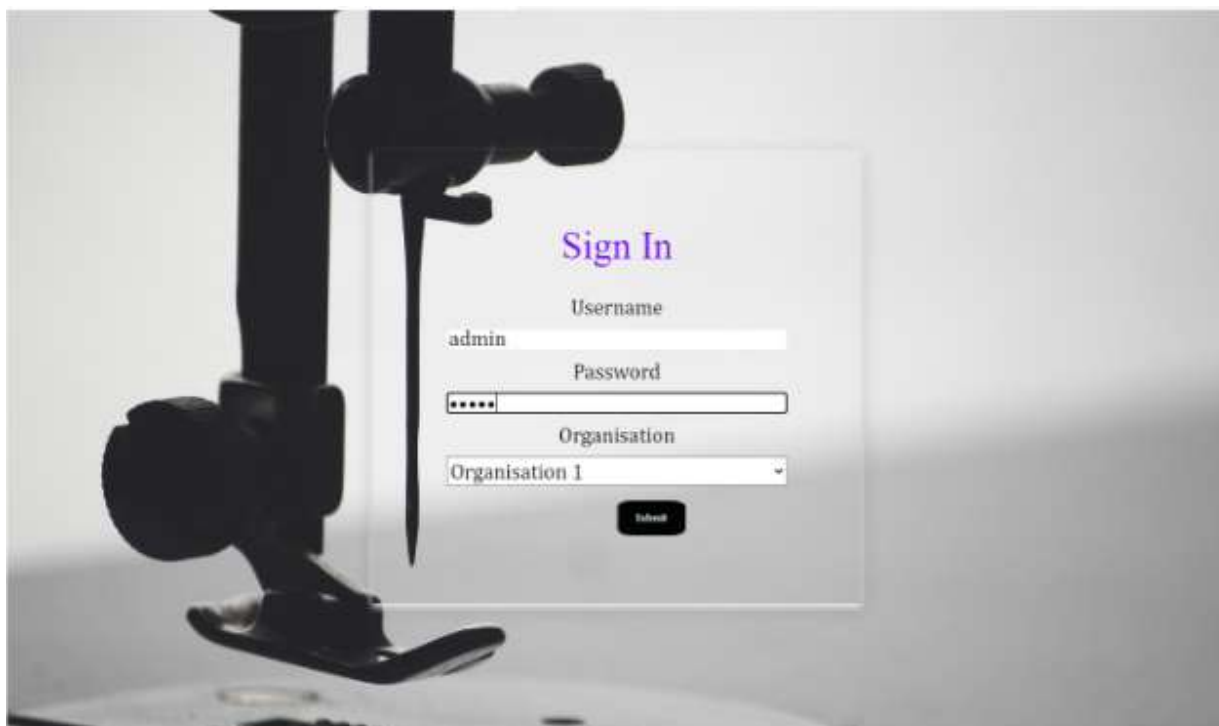


RUNNING DATA BASE

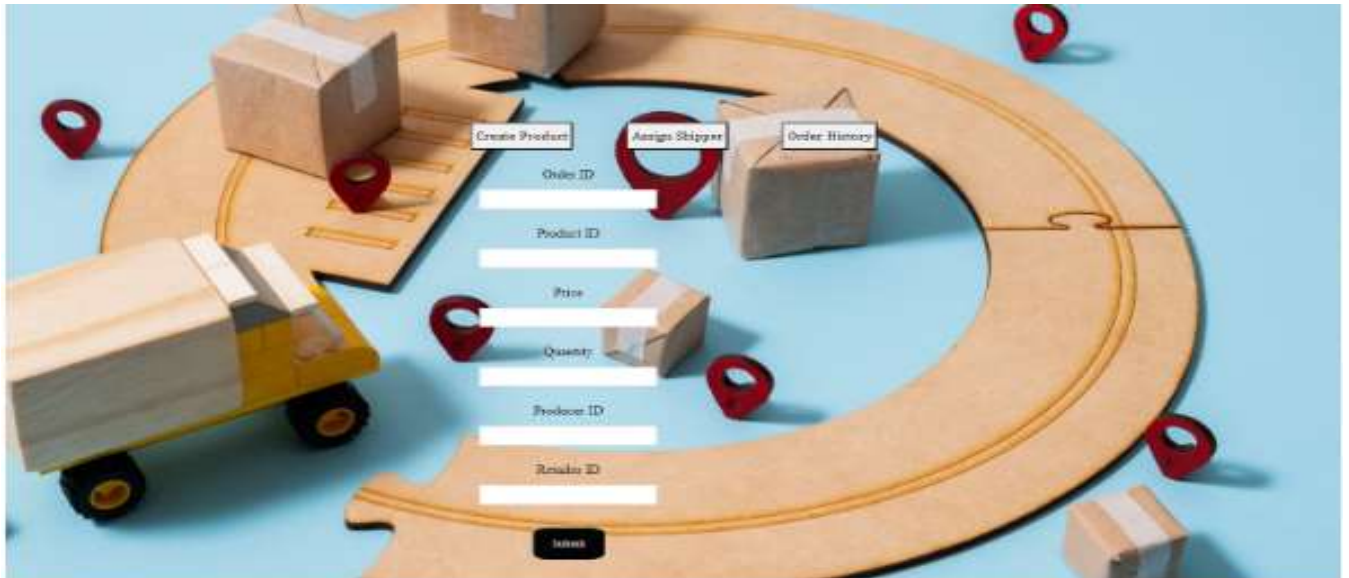
CouchDB is a database management system that falls under the category of NoSQL databases. It is an open-source, document-oriented database that stores data in JSON format. CouchDB is known for its scalability, fault-tolerance, and high availability, making it a popular choice for distributed applications.

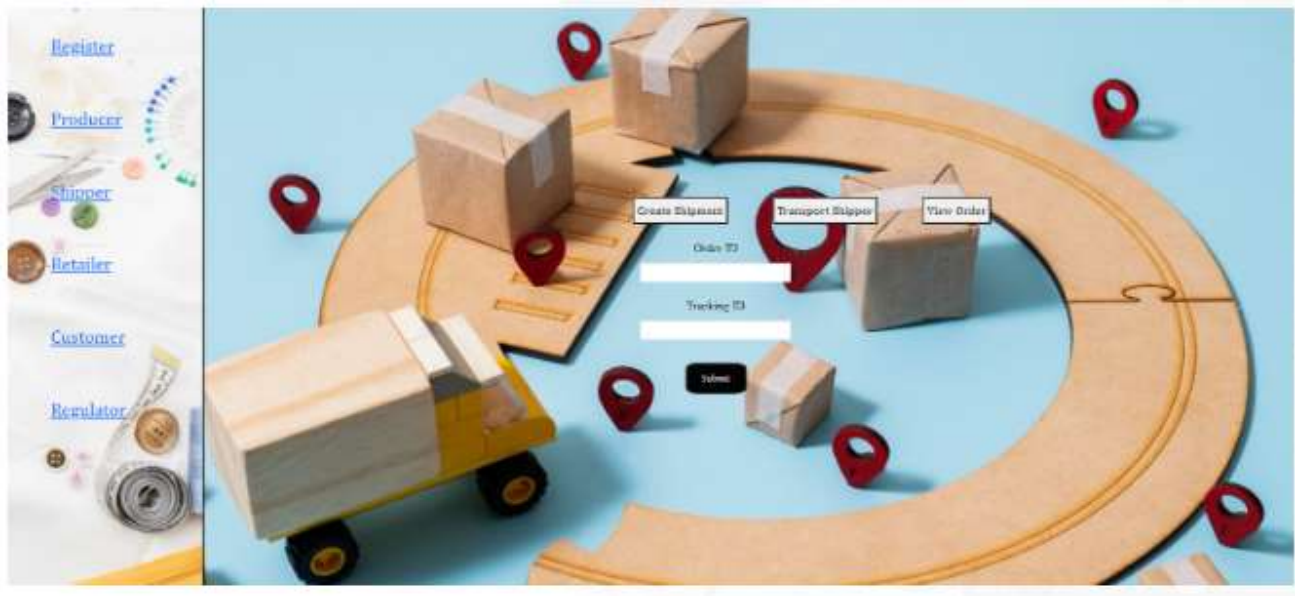


FRONTEND



C-DAC ACTS (Patna)/PG-DFBD







CONCLUSION

In conclusion, the implementation of a Textile Supply Chain Management System using Hyperledger Fabric has several benefits. Hyperledger Fabric provides a secure, permissioned blockchain network that enables secure and transparent communication between all parties in the supply chain. By using a blockchain-based solution, all parties can access the same data in real-time, which can improve communication, reduce errors, and increase efficiency.

In a Textile Supply Chain Management System, using Hyperledger Fabric can help to track the lifecycle of textile products, from raw materials to finished products. By implementing a blockchain-based solution, the system can ensure that all parties in the supply chain can verify the authenticity of the products, trace the origin of raw materials, and monitor the quality of the products throughout the production process.

Additionally, Hyperledger Fabric provides a robust set of tools and APIs that can help to streamline the development and deployment of the Textile Supply Chain Management System. This can save time and reduce the costs associated with developing a custom solution.

Overall, a Textile Supply Chain Management System using Hyperledger Fabric has the potential to revolutionize the textile industry by providing greater transparency, security, and efficiency. By implementing a blockchain-based solution, the industry can improve supply chain management, reduce costs, and enhance customer satisfaction.

REFERENCE

<https://youtu.be/8tVx0r6pgU4>

<https://medium.datadriveninvestor.com/hyperledger-fabric-best-practices-in-production-1-encrypting-state-database-with-chaincode-8369b0bc345a>

<https://medium.com/zeeve/crucial-considerations-before-deploying-hyperledger-fabric-in-the-blockchain-network-in-2023-5ac2cdad1be9>

https://hyperledger-fabric.readthedocs.io/en/release-2.5/test_network.html

<https://hyperledger-fabric.readthedocs.io/en/release-2.5/glossary.html>

https://hyperledger-fabric.readthedocs.io/en/release-2.5/getting_started.html

<https://www.sellbrite.com/blog/quick-guide-product-shipping-everything-need-know/>

<https://images.app.goo.gl/VfRBL3uNTS26cVuU9>