EAS-IPM
Design Document – Pharma Network
VERSION 1.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Nisha M. R	Anaswar G	Vinod Kumar Pai
Role	Developer	Sr. Developer	Associate Director
Signature			
Date	18-10-2019	18-10-2019	

Introduction

1.1 Purpose

The purpose of this document is to provide an explanation of the pharma Network

1.2 Background

Pharma Network provides accurate visibility into inventory levels and shipping of drug across supply chain. Pharmacies receive the drug and can report problem and FDA can withdraw the drug. All participants are notified through the ledger.

1.3 Requirements and Dependencies

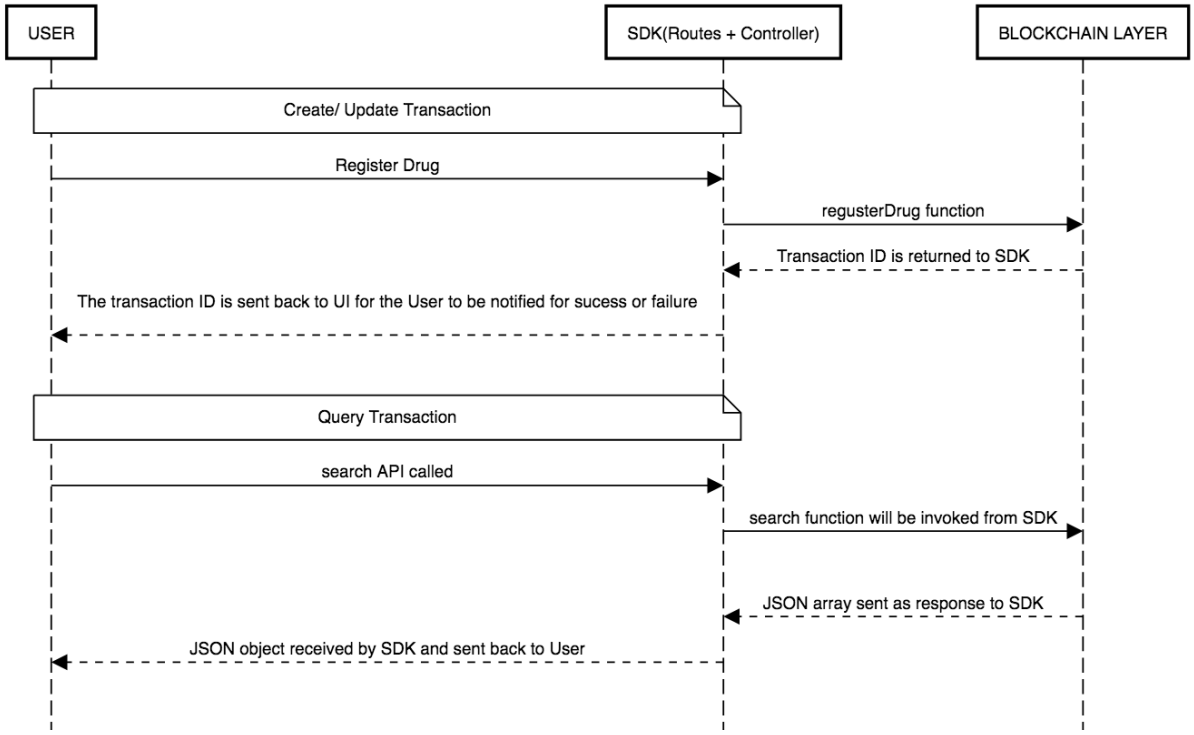
The following were used to build the application.

1) Hyperledger Fabric – 1.4
2) Node JS - v8.11.3
3) GoLang - go1.12
4) Angular JS
5) IDE – Visual Studio Code

1.4 Network Architecture

The application in divided into 3 layers.

1) User Interaction layer
2) SDK Layer
3) Blockchain Layer



1.4.1 User Interaction Layer

The user interaction layer includes a presentation layer. This layer is built using HTML/Angular JS and CSS

1.4.2 SDK Layer

This is the middle/interface layer which is the gateway to the blockchain. The middle layer provides the communication between user interaction layer and the blockchain layer. This layer is built using Node js. Java SDK is also available. The transaction submitted by the user is routed to controller file which is the node layer via a routes.js file. The controller file will have the orderer address and peer address which are participating in the network. The named of installed chaincode, the transaction and the arguments of the transaction is send to the chaincode in the form of request. The transaction is sent using sendTransactionProposal The proposal response will be sent back to sdk by the chaincode and this response can be sent back to UI. For query transaction, the request is sent to chaincode using querybyChaincode.

Controller.js syntax

```
module.exports = (function() {
return{
registerDrug: function(req, res){
    var channel = fabric_client.newChannel('mychannel');
    var peer1 = fabric_client.newPeer('grpc://localhost:7051');
    var peer2 = fabric_client.newPeer('grpc://localhost:8051');
    var peer3 = fabric_client.newPeer('grpc://localhost:9051');
    var peer4 = fabric_client.newPeer('grpc://localhost:10051');
    channel.addPeer(peer1);
    channel.addPeer(peer2);
    channel.addPeer(peer3);
    channel.addPeer(peer4);
    var order = fabric_client.newOrderer('grpc://localhost:7050')
    channel.addOrderer(order);

    var input = req.body
    var drugID = input.drugid;
    var drugName = input.drugname;
    var drugStatus = input.drugstatus;
    var manufacturer = input.manufacturer;
    var components = input.components;
    var argument = [drugID, drugName, manufacturer, drugStatus, components];


     const request = {
        //targets : --- letting this default to the peers assigned to the channel
           chaincodeId: 'pharmachaincode',
           fcn: 'registerDrug',
           args: argument,
           //chainId: 'mychannel',
           txId: tx_id
       };
    // send the transaction proposal to the peers
    return channel.sendTransactionProposal(request);

}
}
});
```

Here peer1 and peer 2 belongs to org1 and peer3 and peer4 belongs to org2.We have to specify the peer to which is acting as committing peer.

```
//let event_hub = fabric_client.newEventHub();
// event_hub.setPeerAddr('grpc://localhost:7051');
let event_hub = channel.newChannelEventHub('localhost:7051');
```

For search transaction, the target peer has to be specified along with request.

```
    const request = {
        targets : [peer1],
        chaincodeId: 'pharmachaincode',
        txId: tx_id,
        fcn: 'search',
        args: [ID]
    };
    console.log(request)
    // send the query proposal to the peer
    return channel.queryByChaincode(request);
}).then((query_responses) => {
```

1.4.3 Blockchain Layer

In the blockchain layer we have the chaincode or smart contract. The chaincode is written in Golang. The call to chaincode is done through SDK. The explanation of chaincode and the functions implemented are explained in later section.

1.5 Application Participants

1) Manufacturer
2) Distributor
3) Pharmacy
4) FDA (Food and Drug Administration)

1.6 Application Description

There are four participants in the network as mentioned in Application participants section.

The manufacturer registers a drug with the FDA. The details of the drug will be available in the ledger.

The pharmacy will create an order for a particular drug.

The drug will be transported by distributor to the pharmacy which placed the order. The pharmacy receives the order from the distributor and sells to the customer.

Temperature is recorded at various points of transportation of drug and the following parameters are recorded by using an IoT sensor.

1) Temperature
2) Moisture

3) Shake

If the temperature is recorded greater than 50 at any point of time, the order will be rejected and will not be allowed to ship the order further. If the order is at pharmacy the drug from rejected order will not be allowed to sell. FDA can completely withdraw a drug if found not good for consumption.

In each of the organizations, one pair is defined as anchor peer. A channel named “mychannel” is created and the all the peers are added to the channel. There is a script file to start the hyperledger fabric network and to install the chaincode. While running the script file the peer, chaincode, orderer and CA containers are created.

The commands to install, instantiate and invoke chaincode needs to be specified in the shell script file in the first network of fabric-samples (utils.sh).

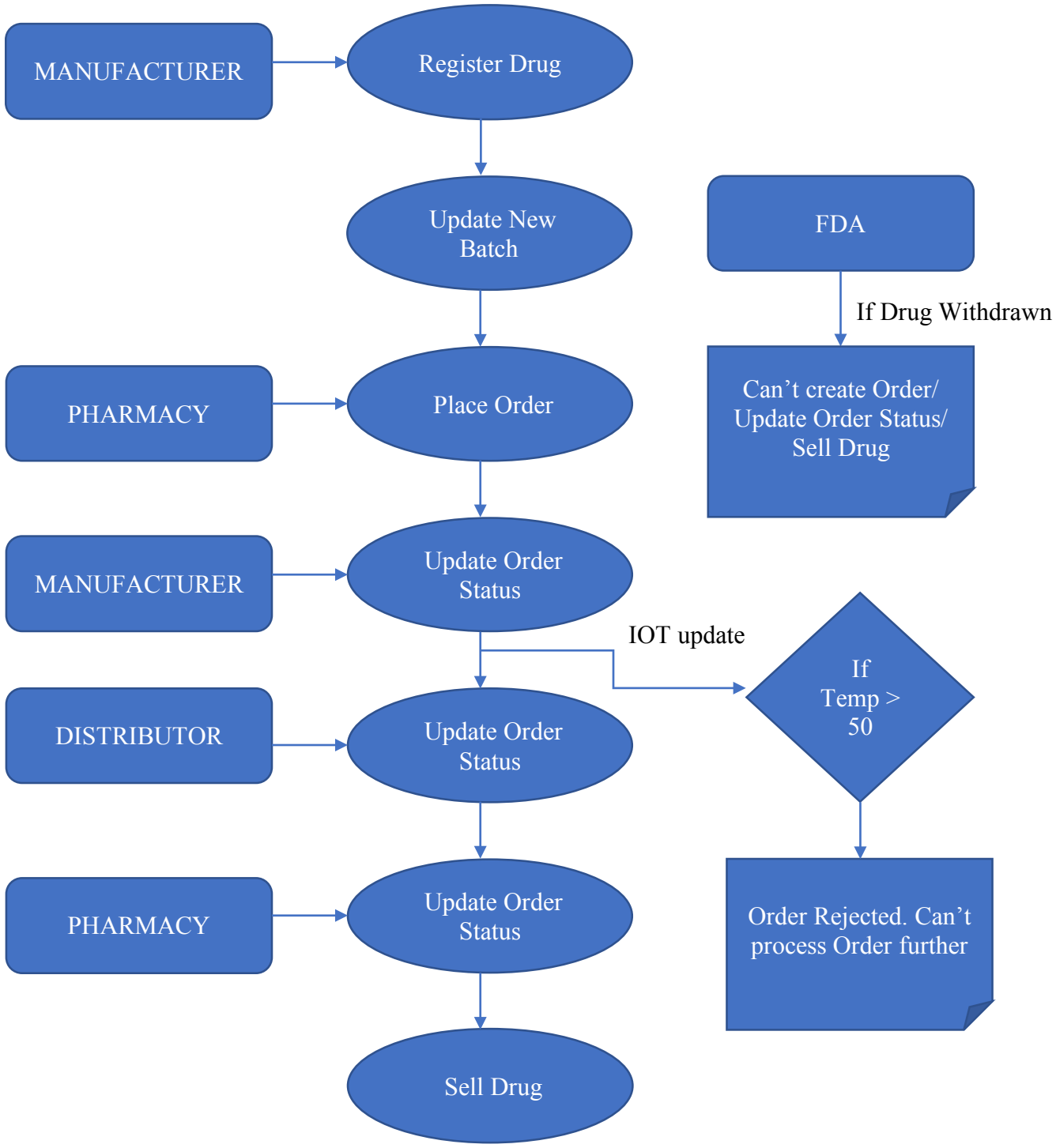
Syntax:

```
peer chaincode install -n pharmachaincode -v ${VERSION} -l ${LANGUAGE} -p ${CC_SRC_PATH} >&log.txt
```

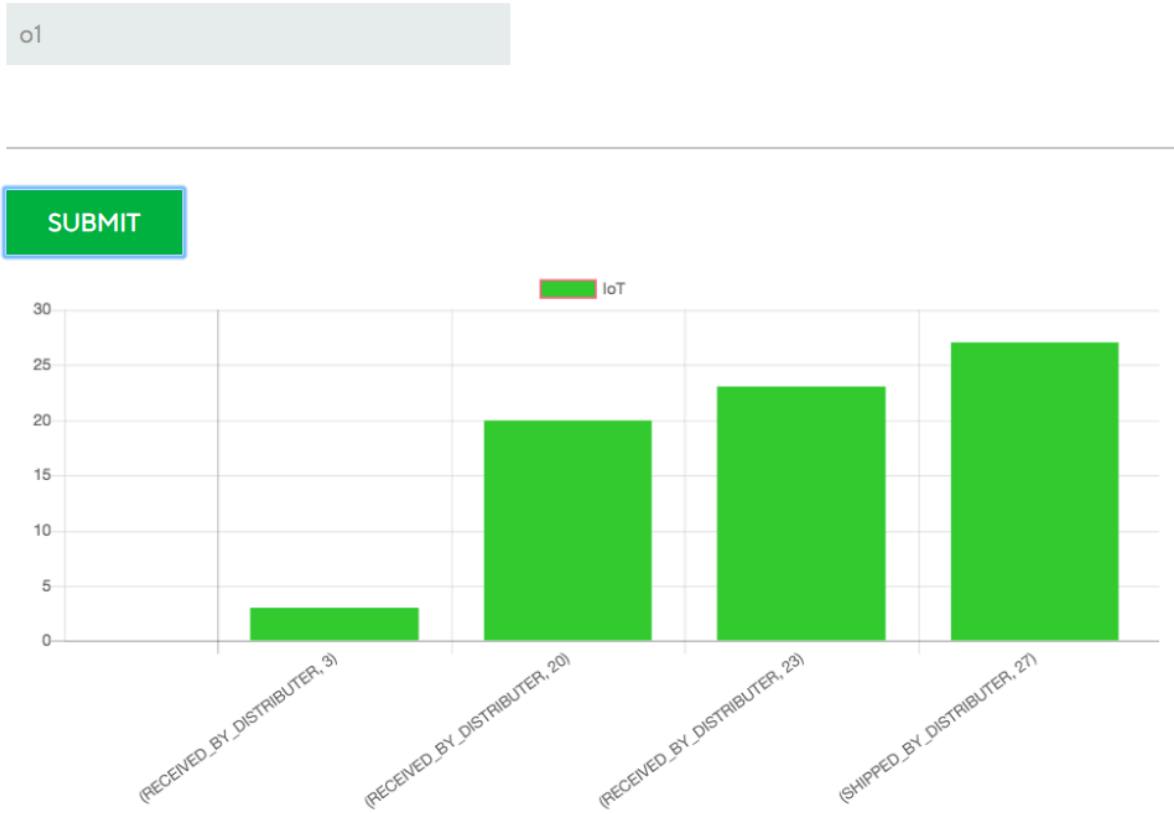
```
 peer chaincode instantiate -o orderer.example.com:7050 -C $CHANNEL_NAME -n pharmachaincode -l ${LANGUAGE} -v ${VERSION} -c '{"Args":["init"]}' -P "OR ('Org1MSP.member','Org2MSP.member')" >&log.txt
```

```
peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n pharmachaincode $PEER_CONN_PARMS -c '{"Args":["registerDrug","Drug02","Drugname01","drug01manu","Available","[c1,c2]"]}' >&log.txt
```

1.7 Flow Diagram



IoT can update temperature at any point of order transportation. All the temperatures will be recorded and can be seen as a graph.



1.8 Chaincode

The chaincode is written in Golang. The packages needed for implementation are imported from "Shim". The important functions defined for chaincode are

a) Init

Init function is called during initialization or update done on the chaincode. Init function takes ChaincodeStubInterface as parameter. When we call this function from CLI, it responds back with the peer response.

```
//Init - initialization of chaincode
func (t *PharmaChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
  return shim.Success(nil)
}
```

b) Invoke

All the logic implementation functions are written here. These implementation functions are called transactions. There are insert, update and query transactions. The transaction list is explained later.

Syntax:

```
func (t *PharmaChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
   function, args := stub.GetFunctionAndParameters()
   if function == "registerDrug" {
      return t.registerDrug(stub, args)
   }
   return shim.Error("Error invoking function")
}
```

1.8.1 Assets

The following assets are defined in the chaincode.

1 Drug

ID – DrugID

The details of the drug created are stored in drug asset. This can be identified using Drug Id.

2 Order

ID – OrderID

The order when created by pharmacy will be stored here. The order asset will get updated for each change of shipment status.

3 Batch

ID – BatchCode

The drug quantity is updated in batches. When updating new batch, this asset is created. The problem if reported on the drug will be of particular batch.

1.8.2 Transactions

1) registerDrug

New drug is created by the manufacturer for a particular drug ID. The drug asset is created during this transaction.

2) updateNewBatch

The drug has to be updated with certain quantity. The complete quantity of drug is updated in batches. The asset batch is created and drug is updated during this transaction.

3) placeOrder

Order is placed by pharmacy with unique order ID. The order will be created within the available quantity of drug. Order asset is created during this transaction.

4) updateOrderStatus

Manufacturer, distributor and pharmacy invoke this transaction to transport or receive the order placed by pharmacy. All the assets will be updated during invoking this transaction.

5) updateDrugStatus

FDA invokes this transaction to withdraw a drug if any problem is found in the drug. The drug will not be further processed in future. And no order can be created for that drug

6) reportProblem

Pharmacy can report problem with the order placed by them. When this transaction is invoked the pharmacy can conduct audit with the supplier who dispatched the order.

7) temperatureUpdate

The temperature update is done by IoT device for a specific order at any point of time. If the temperature read on the device is greater than **50(threshold limit)**, then the order will be rejected. The temperature history is displayed to user in the form of chart to check the temperatures at each time of updating.

8) Search

This transaction is used to query the assets using their unique ID. Target peer will be specified at the time of querying.

1.9 Running Network

For running the application, we have to start the fabric network, install the chaincode and then the node SDK is started. The following steps have to be followed.

1) Navigate to the folder which contain the shell script for starting the fabric. Here there is a startFabric.sh script. When we run this script, it will call the byfn.sh script in the network folder.

   ./startFabric.sh

If this script is run successfully, the following result will be displayed.

```
========== All GOOD, BYFN execution completed ===========

 _____   _   _   ____
| ____| | \ | | |  _ \
|  _|   |  \| | | | | |
| |___  | |\  | | |_| |
|_____| |_| \_| |____/

Total execution time : 451 secs ...

Start with the registerAdmin.js, then registerUser.js, then server.js
```

2) We can see the list if containers that are up and running using the following command.

docker ps –a

```
AIM00110:PHARMA-APP 549121$ docker ps -a
CONTAINER ID        IMAGE                                                                                                                                COMMAND
609eec84afce        dev-peer0.org2.example.com-pharmachaincode-1.0-e53a05ab747a6222867f96bc4ee32dfd3516ea2f1b9e4c1ea8de0fef9ba5af32                       "chaincode -peer.add…"
86e29f391d85        dev-peer1.org1.example.com-pharmachaincode-1.0-c7fe57587bd7d2115969647930fd8ad00584158db268c55447fbf8c107fcf630                       "chaincode -peer.add…"
aa4128d2c818        dev-peer0.org1.example.com-pharmachaincode-1.0-a75684b40e0ae944f0a9096e0913207c4e3da24c7cdc530ff70f3b4eee8eddc3                       "chaincode -peer.add…"
34792b763078        dev-peer1.org2.example.com-pharmachaincode-1.0-bf3dd5adab6d58b78e5645749b00ee6e024e11775dc5f4932066407373aaaaec                       "chaincode -peer.add…"
f0ac6050e6da        hyperledger/fabric-tools:latest                                                                                                      "/bin/bash"
16ea423c615b        hyperledger/fabric-orderer:latest                                                                                                    "orderer"
999680925c0f        hyperledger/fabric-ca:latest                                                                                                         "sh -c 'fabric-ca-se…"
7a4e762abf84        hyperledger/fabric-ca:latest                                                                                                         "sh -c 'fabric-ca-se…"
7fdcb481b7ee        hyperledger/fabric-peer:latest                                                                                                       "peer node start"
531cede17bb8        hyperledger/fabric-peer:latest                                                                                                       "peer node start"
e839ef6725c8        hyperledger/fabric-peer:latest                                                                                                       "peer node start"
a5ffa1a38e2b        hyperledger/fabric-peer:latest                                                                                                       "peer node start"
```

Now the chaincode is installed and instantiated in all the peers of the channel.

3) Node modules

The required node modules need to be installed. This is done using command

npm install

This will install the packages that are predefined in package.json. If any error specifying package is not available, we can install that using command npm install.

4) **registerAdmin.js** and **registerUser.js**

The admin and users have to be enrolled for each ca. In the network since this is multi org, there are two ca containers and using the certificate file of ca we have to enroll admin and user.

For enrolling admin

node registerAdmin.js

For enrolling user

node registerUser.js

This will create a **hfc-key-store** in the SDK folder and create the user and admin. This is later used by controller when committing a transaction.

5) Now we have to start the node app

Go to the root folder and hit the following command

npm start OR **node server.js**

The following will be displayed if the app is started successfully.

```
AIM00110:PHARMA-APP 549121$ npm start

> Fabric-ElectriCharge-app@1.0.1 start /Users/549121/Desktop/PHARMANETWORK/PHARMA-APP
> node server.js

Live on port: 8000
```

We can hit the application using url: http://localhost:8000/