Table of Contents

Abstract:

- The main intention of this project is to show how a drug-tracking supply chain can be used and efficiently delivered through a permissioned blockchain framework. A comprehensive presentation was shown to visualize the agenda and goals of this project (also attached as part of this submission).
- To re-iterate, blockchain can be used to enforce secure and efficient movement of drugs from the manufacturer to the customer. This industry involves movement of delicate information and is prone to counterfeiting, a lot of paperwork, time consuming processes, and this can be addressed through a permissioned blockchain with smart contract technology.
- We use Hyperledger Fabric as the blockchain framework and address this use-case by means of a full-stack web application. More on how the application works, the technological components involved, are listed below along with a video demo link.

Hyperledger Fabric for drug-tracking:

Hyperledger Fabric is IBM/Linux's open-source framework for developing enterprise level blockchain applications. The main features of Hyperledger Fabric that made us go for this framework are,

- Permissioned membership
- Rich queries over an immutable distributed ledger
- Channel technology for confidential transactions
- Endorsement policy for transaction validation
- Modular architecture with easy plug and play components
- Protection of sensitive data using digital keys (MSP)
- Novel Architecture (Execute-Order-Validate)
- Program friendly Chaincode (smart contract)

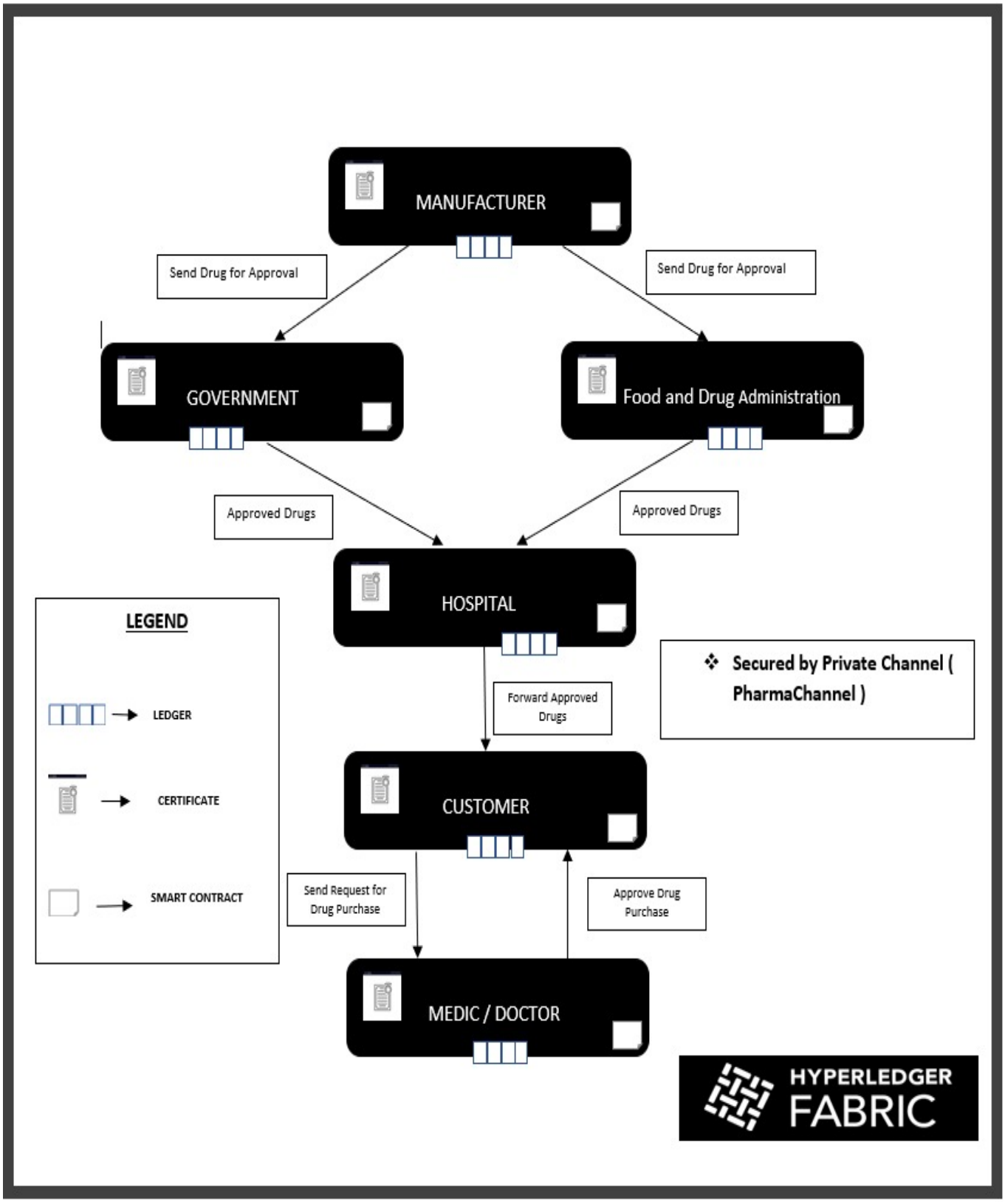
Thus, for the drug tracking supply chain,

- ✓ Blockchain can enforce safe drug production and anomalies/errors can be traced back to the source. **(Fabric comes with channels that have decentralized ledgers with easy and rich queries)**
- ✓ This is an industry that transacts delicate information, prone to counterfeiting, illegal manufacture of drugs, maintaining proper drug standards etc. and blockchain can help address these issues. **(Fabric is permissioned and only select parties can transact with cryptographic MSP)**
- ✓ In general, the supply-chain involved with drug-tracking is paperwork heavy, involves lot of manual stuff, labor costs are involved, and processing time takes few days – Fabric can speed up all this **(Fabric is fast, chaincode can automate document verification in near real-time)**

Basic flow for understanding:

- The below diagram represents the flow of drug-related information and architecture among the peer actors in the network. Every peer actor as you can see contains a copy of the blockchain ledger itself, smart contracts for endorsement, and MSP/cryptographic material for authorizing and encrypting transactions in the blockchain.
- The peers are connected to a secure channel for enforcing privacy (if you recall, you can also configure multiple channels in a network, for different interactions).
- Suppose the manufacturer, enters drug related information, and clicks on add drug, the information (drug related fields like name, chemical etc.) are added as arguments in a Transaction proposal request.
- This request is sent to all the other peers (peers required to endorse the transaction).
- Once all the peers (which hold the smart contract) execute the contract and endorse the transaction, the orderer combines it into a block and adds it to the blockchain. This is distributed to all the peers in the network.
- This happens similarly for other transactions, like drug approval or rejection. (Code snippets and screenshots are shown in the upcoming sections)
- The **Endorsement policy** for this use-case states that all the peers need to execute the transaction and agree on the results. This is defined during chaincode instantiation. This forms the consensus part for this application.

You can refer to the video link, for detailed explanation of the flow.


MANUFACTURER
Send Drug for Approval
Send Drug for Approval
GOVERNMENT
Food and Drug Administration
Approved Drugs
Approved Drugs
HOSPITAL
LEGEND
LEDGER
CERTIFICATE
SMART CONTRACT
Secured by Private Channel (PharmaChannel)
Forward Approved Drugs
CUSTOMER
Send Request for Drug Purchase
Approve Drug Purchase
MEDIC / DOCTOR
HYPERLEDGER FABRIC

Development Environment/Technology Involved:

- Hyperledger Fabric Binaries/Samples (v1.4.4)
- GoLang (Smart Contract)
- Docker
- Spring MVC Framework, Fabric Java SDK (Backend)
- HTML, CSS, JavaScript (Frontend)

For setup instructions refer the UserManual.pdf document provided.

Sample Code Snippets:

1) To access the blockchain network which is started, we first need to set up the Fabric Java SDK by providing the crypto material, setting the User context (in our case, every peer has admin capabilities) and then accessing the channel, the peers and orderer through their request ports(via grpc calls). A sample code is given below. We first create a Fabric Client Object – hfClient. We then access the channel and the peers by providing their certificates. We persist this context for the entire session.

```
Properties peer3Properties = new Properties();
peer3Properties.setProperty("pemFile",
        currentUsersHomeDir+"\\crypto-config\\peerOrganizations\\manufacturer.state.com\\peers\\peer0.manufacturer.state.com\\tls\\server.crt");
peer3Properties.setProperty("trustServerCertificate", "true"); // testing environment only NOT FOR PRODUCTION!
peer3Properties.setProperty("hostnameOverride", "peer0.manufacturer.state.com");
peer3Properties.setProperty("sslProvider", "openSSL");
peer3Properties.setProperty("negotiationType", "TLS");
peer3Properties.put("grpc.NettyChannelBuilderOption.maxInboundMessageSize", 9000000);
Peer peer3 = hfClient.newPeer("peer0.manufacturer.state.com", "grpcs://localhost:9051", peer3Properties);
```

2) For a **createDrug** event from the user interface,
Suppose the manufacturer, enters drug details and hits the createDrug button, an Ajax call is sent along with the data to the Java Controller method as shown below.

```
var id = n+1;
var name = $("#name").val();
var type = $("#type").val();
var num = $("#num").val();
var chemName = $("#chemName").val();
var nat = $("#nat").val();
var sup = $("#sup").val();
var ulab = $("#ulab").val();
var se = $("#se").val();
var strg = $("#strg").val();

$.ajax({
    url : "createDrug",
    data : ({
        id: id,
        name : name,
        type : type,
        num : num,
        chemName : chemName,
        nat : nat,
        sup : sup,
        ulab : ulab,
        se : se,
        strg : strg
    }),
    success : function(data) {
```

These parameters (the drug details) are added as a TransactionProposalRequest, which is a method of the Fabric Provided Java SDK. We then get the channel instance (which we setup before) and

then call the chaincode (the smart contract), set the function parameters and send the request to all the peers. The peers send back the transaction execution response in a ProposalResponse object. We return back the status code which is returned and given as a success alert in the web interface

```
public int createDrug(String id, String name, String type, String num, String chemName, String nat, String sup,
        String ulab, String se, String strg) throws ProposalException, InvalidArgumentException {
    // TODO Auto-generated method stub
    TransactionProposalRequest req = BlockChainHFClient.getInstance().getClient().newTransactionProposalRequest();
    Channel channel = BlockChainHFClient.getInstance().getCh();
    ChaincodeID cid = ChaincodeID.newBuilder().setName("drugtracker_chaincode").build();
    req.setChaincodeID(cid);
    req.setFcn("createDrug");
    req.setArgs(new String[] { "Drug" + id, name, type, num, chemName, nat, sup, ulab, se, strg, "Approval Pending",
            "Aprroval Pending" });

    Collection<ProposalResponse> resps = channel.sendTransactionProposal(req);
    int status = 0;
    channel.sendTransaction(resps);
    for (ProposalResponse pres : resps) {
        status = pres.getChaincodeActionResponseStatus();
    }

    return status;
}
```

Go code snippet for createDrug() – the smart contract with a PutState() for addition to the ledger

```
func (s *SmartContract) createDrug(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

    if len(args) != 12 {
        return shim.Error("Incorrect number of arguments. Expecting 12")
    }

    var drug = Drug{Name: args[1], Type: args[2], Amount: args[3], Chemname: args[4], Nature: args[5],
                    Supplier: args[6], Ulabel: args[7], Sides: args[8], Storage: args[9], FdaValid: args[10], GovtValid: args[11] }


    drugAsBytes, _ := json.Marshal(drug)
    APIstub.PutState(args[0], drugAsBytes)

    return shim.Success(nil)
}
```

3) Sample snippet for queryDrugs – when we click the get Drugs button, the same scenario as for create drugs happens, except here instead of a TransactionProposalRequest, we have a QueryByChaincodeRequest object.

```java
public String queryAllDrugs() throws InvalidArgumentException, ProposalException {
    QueryByChaincodeRequest request = BlockChainHFClient.getInstance().getClient().newQueryProposalRequest();
    Channel channel = BlockChainHFClient.getInstance().getCh();
    ChaincodeID ccid = ChaincodeID.newBuilder().setName("drugtracker_chaincode").build();
    request.setChaincodeID(ccid);
    request.setFcn("queryAllDrugs");
    String args = null;
    if (args != null)
        request.setArgs(args);

    Collection<ProposalResponse> response = channel.queryByChaincode(request,
            BlockChainHFClient.getInstance().getAdminPeer());
    String stringResponse = null;
    for (ProposalResponse pres : response) {
        stringResponse = new String(pres.getChaincodeActionResponsePayload());
    }
    return stringResponse;
}
```

```go
func (s *SmartContract) queryAllDrugs(APIstub shim.ChaincodeStubInterface) sc.Response {

    startKey := "Drug0"
    endKey := "Drug999"

    resultsIterator, err := APIstub.GetStateByRange(startKey, endKey)
    if err != nil {
        return shim.Error(err.Error())
    }
    defer resultsIterator.Close()

    // buffer is a JSON array containing QueryResults
    var buffer bytes.Buffer
    buffer.WriteString("[")

    bArrayMemberAlreadyWritten := false
    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }
        // Add a comma before array members, suppress it for the first array member
        if bArrayMemberAlreadyWritten == true {
            buffer.WriteString(",")
        }
        buffer.WriteString("{\"Key\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResponse.Key)
        buffer.WriteString("\"")

        buffer.WriteString(", \"Record\":")
        // Record is a JSON object, so we write as-is
        buffer.WriteString(string(queryResponse.Value))
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
    }
    buffer.WriteString("]")

    fmt.Printf("- queryAllDrugs:\n%s\n", buffer.String())

    return shim.Success(buffer.Bytes())
}
```

The above smart contract function queryAllDrugs() as you can see gets the state from the blockchain ledger using the getStateByRange() method for every drugId.

4) The below snippets represent a sample Docker configuration template for a peer (configuration stipulated by Fabric) and their environment variable configurations. We do this similarly for the 6 peers in our network, the orderer and then start the configuration using Docker commands.

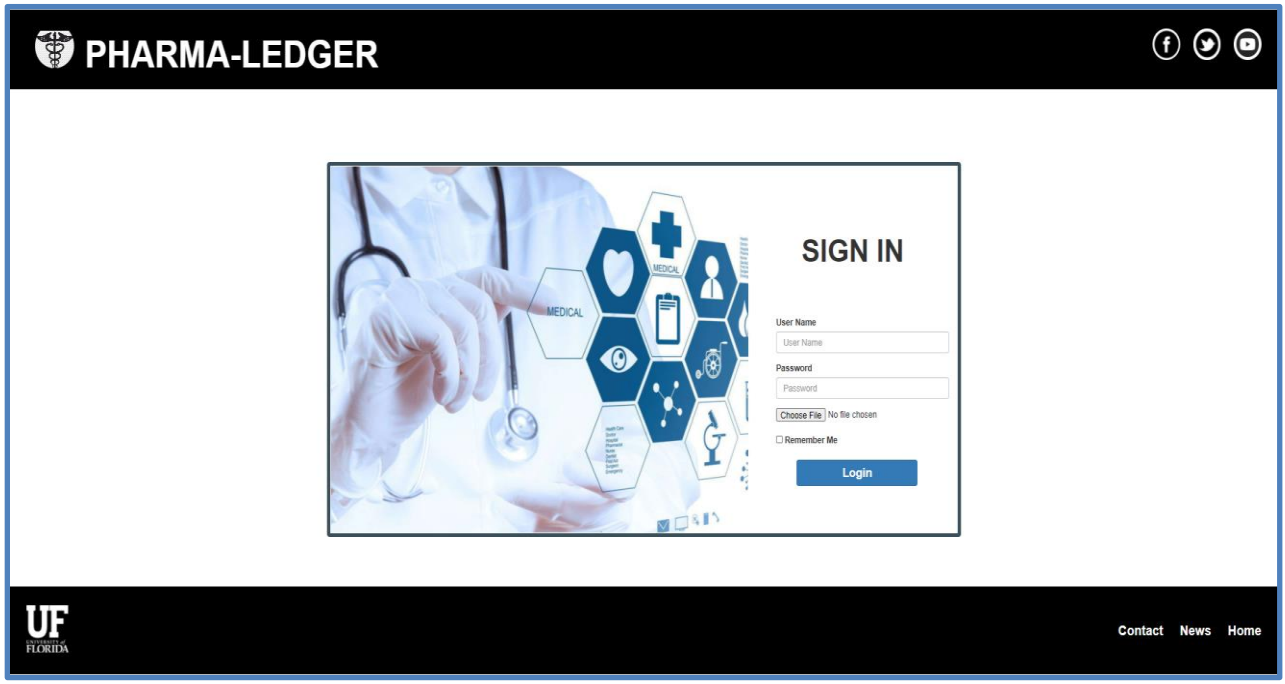
```
peer0.manufacturer.state.com:
  container_name: peer0.manufacturer.state.com
  extends:
    file: peer-base.yaml
    service: peer-base
  environment:
    - CORE_PEER_ID=peer0.manufacturer.state.com
    - CORE_PEER_ADDRESS=peer0.manufacturer.state.com:9051
    - CORE_PEER_LISTENADDRESS=0.0.0.0:9051
    - CORE_PEER_CHAINCODEADDRESS=peer0.manufacturer.state.com:9052
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:9052
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.manufacturer.state.com:9051
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.manufacturer.state.com:7051
    - CORE_PEER_LOCALMSPID=manufacturerMSP
  volumes:
      - /var/run/:/host/var/run/
      - ../crypto-config/peerOrganizations/manufacturer.state.com/peers/peer0.manufacturer.state.com/msp:/etc/hyperledger/fabric/msp
      - ../crypto-config/peerOrganizations/manufacturer.state.com/peers/peer0.manufacturer.state.com/tls:/etc/hyperledger/fabric/tls
      - peer0.manufacturer.state.com:/var/hyperledger/production

  ports:
    - 9051:9051
```

```
peer-base:
  image: hyperledger/fabric-peer:$IMAGE_TAG
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_scriptToRun
    - FABRIC_LOGGING_SPEC=INFO
    #- FABRIC_LOGGING_SPEC=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  command: peer node start
```

Screenshots:

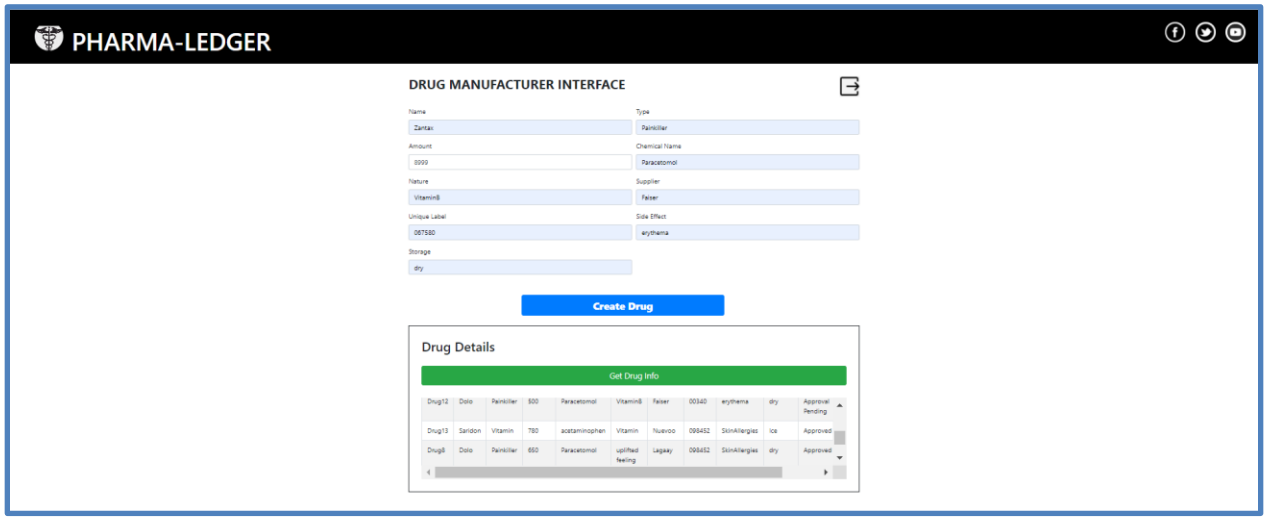
Peer Login Page

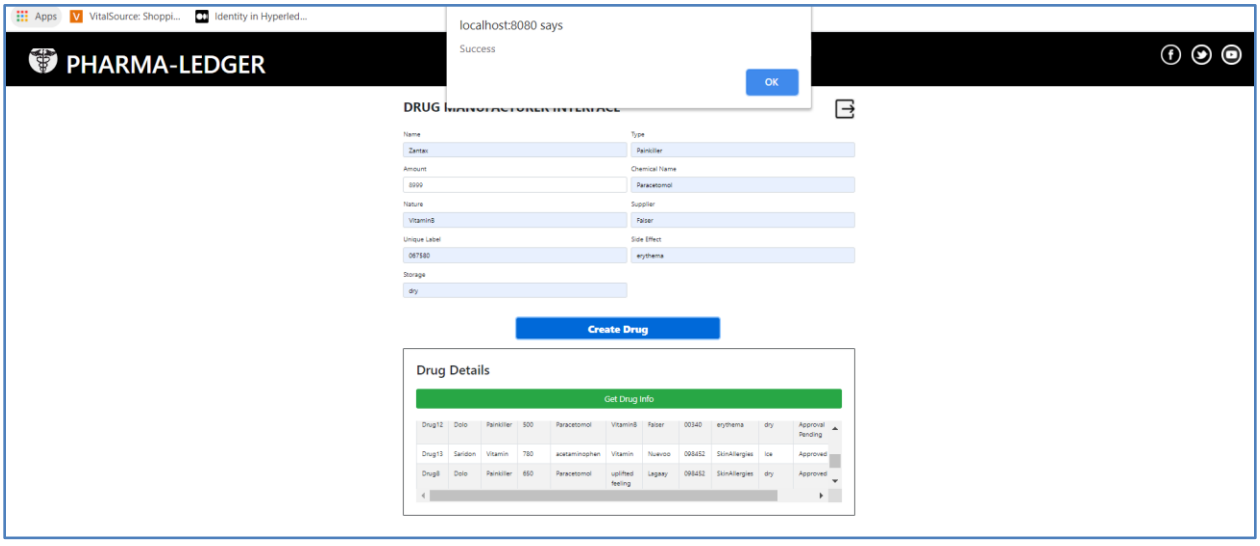
Peer login by giving the appropriate certificate file for authentication and signed transactions in the blockchain network. (Login credentials are given in the User setup manual)



Manufacturer Peer Interface

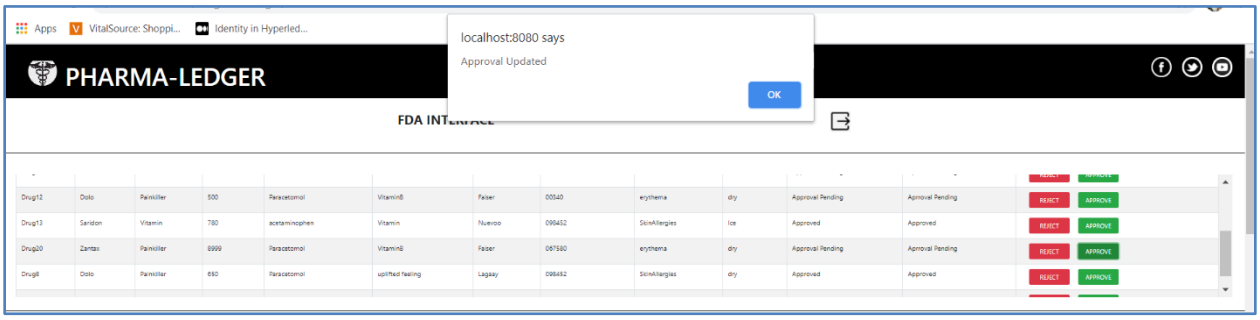
Interface for the manufacturer, where the drug is created, added to the ledger and can be queried. The drugs are to be approved by the FDA and government entity for retail rights. REMEMBER that every button click is a transaction/query call to the blockchain.

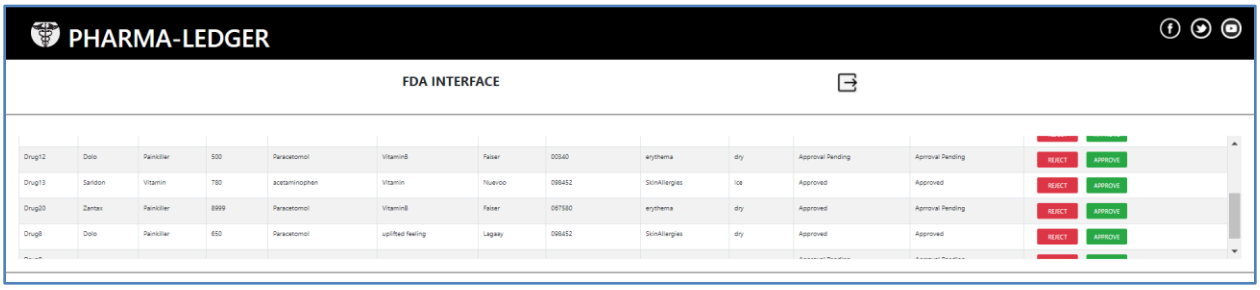




Food And Drug Administration (FDA) Peer Interface

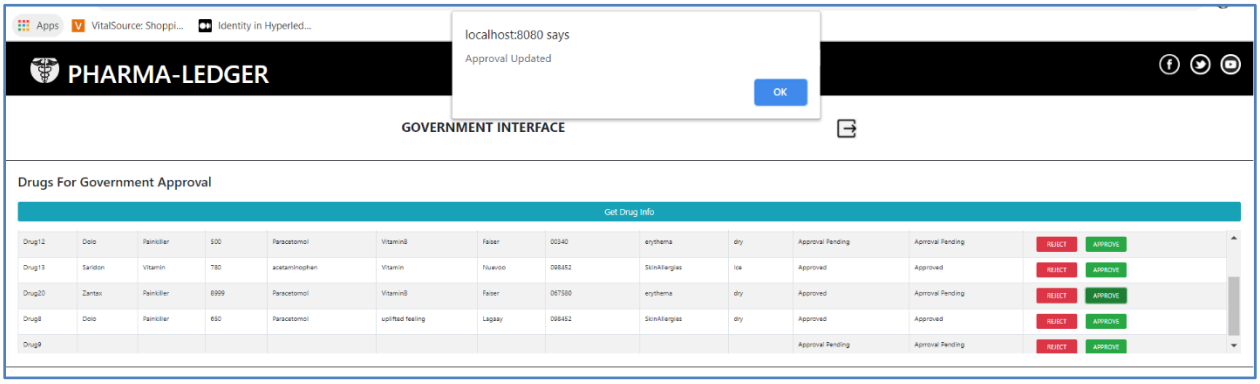
Once the drug is added by the manufacturer, the FDA can access the drug details from the ledger and approve or reject the drug.

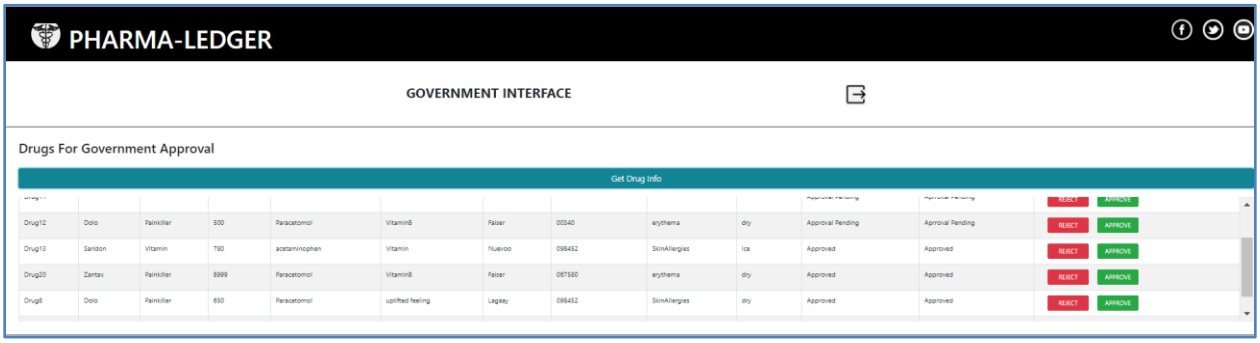




Government Peer Interface

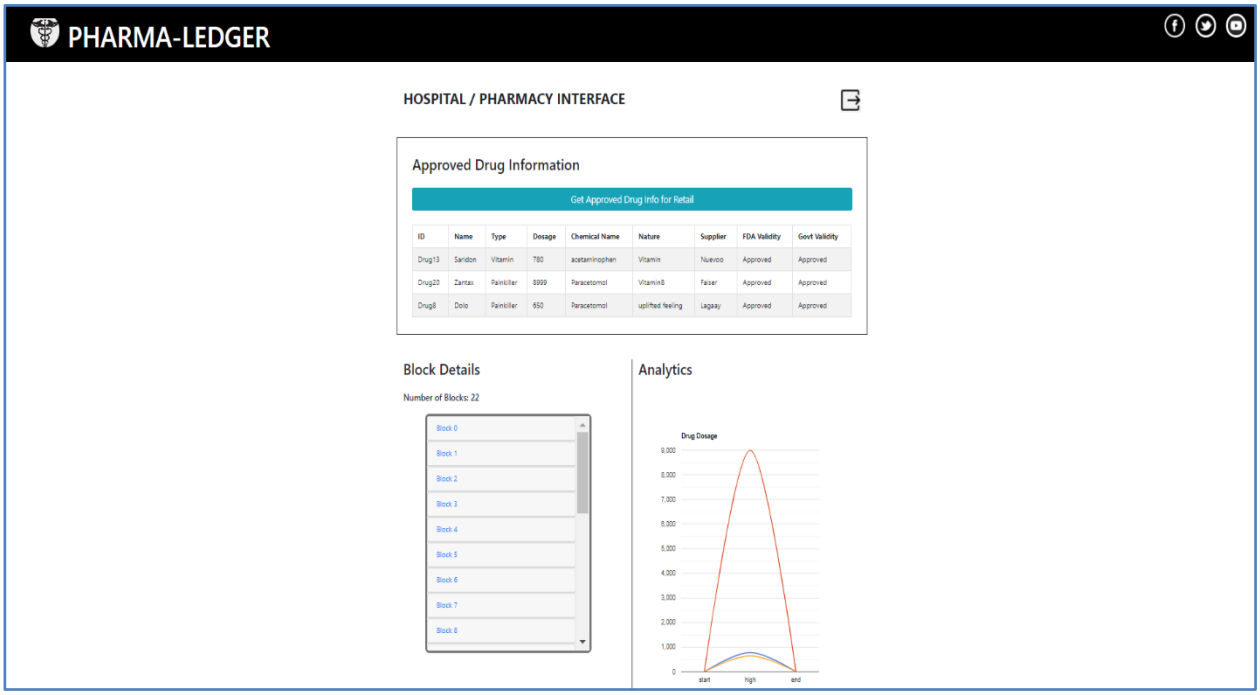
Similar to the FDA, an entity in the government should also approve/reject the drug.





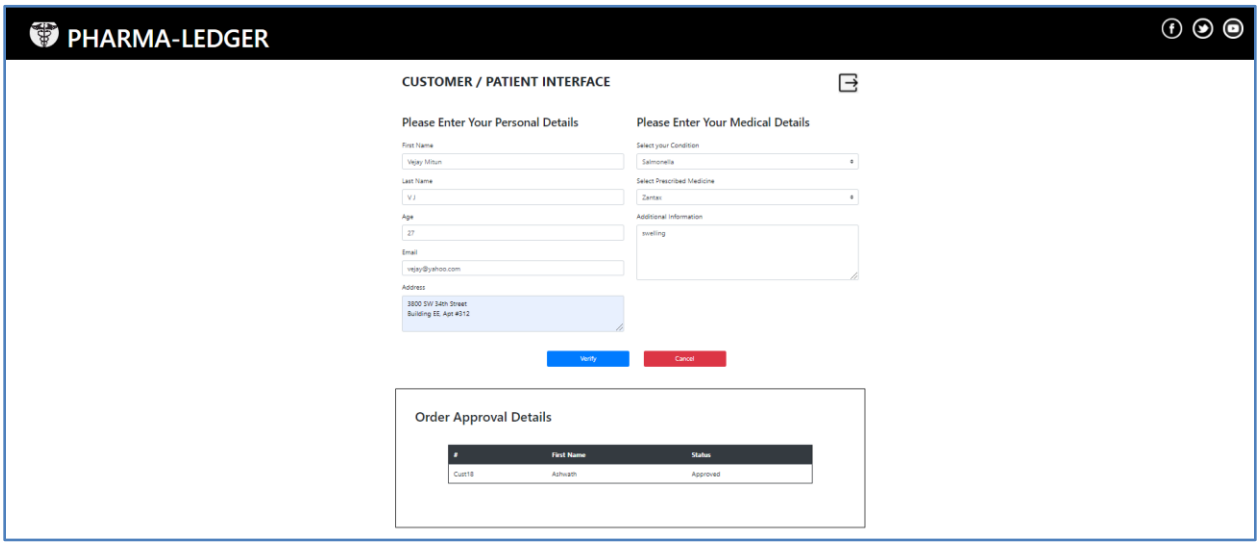
Hospital / Pharmacy Peer Interface

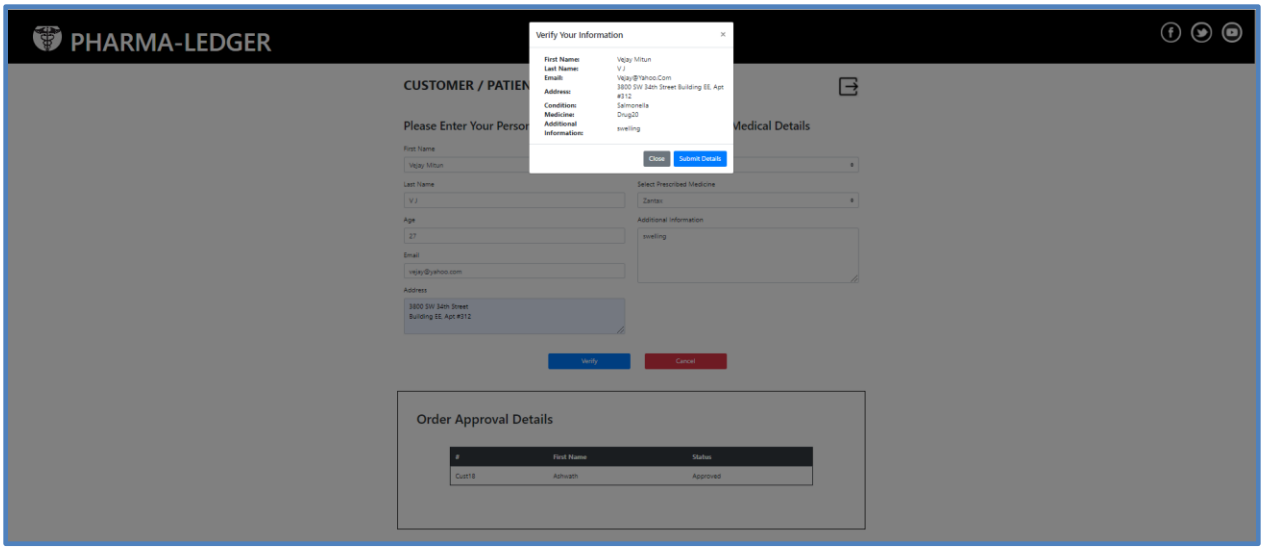
Once the drug is approved by both the FDA and the government entity (only then), the drugs are approved for retail and can be queried at the hospital or pharmacy. You can see that FDA and Govt Validity indicate ‘Approved’. A sample chart is shown for Drug and dosage analysis. This is to show that since data is coming at near real-time, it can be used for analytics purposes as well.



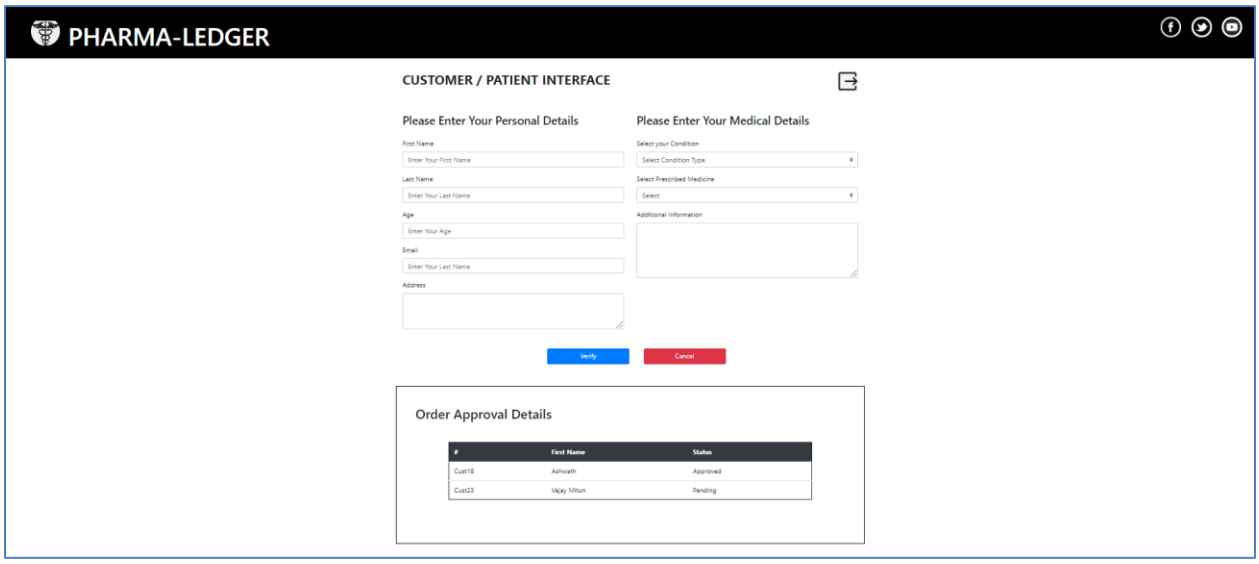
Customer / Patient Peer Interface

This interface is for retail transparency. When the customer purchases a drug based on his condition, the details are added to the ledger, and sent to the medic/doctor for approval. This is to make sure that the drugs are not bought with bad/illegal prescriptions.



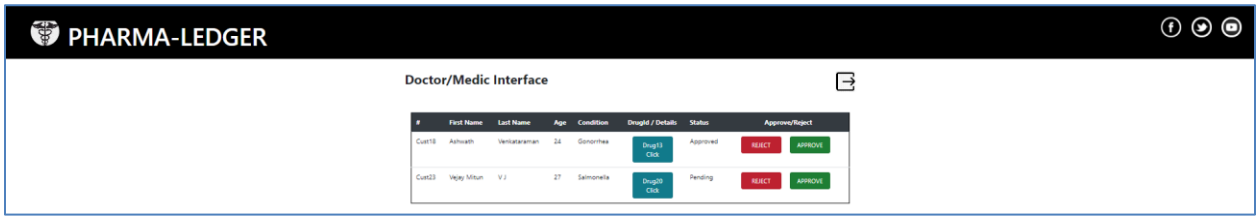


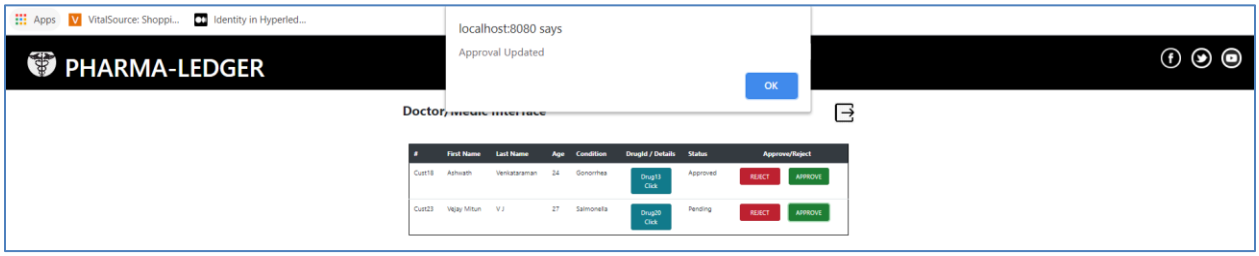
As you can see, the approval details are pending from the medic/doctor's end.

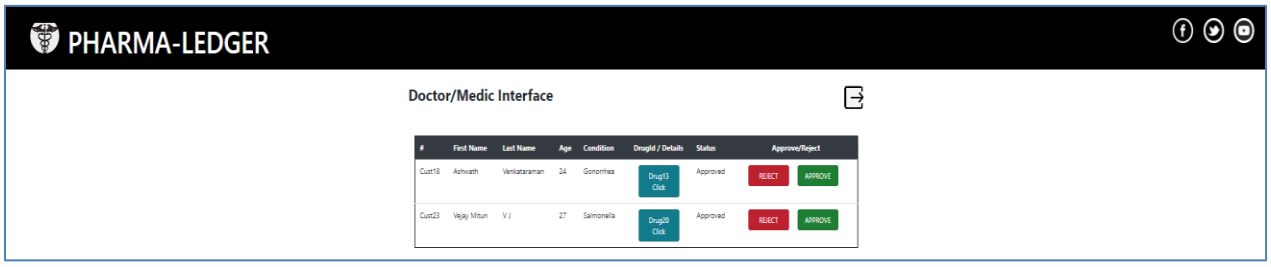


Doctor / Medic Peer Interface

The doctor gets the customer details, along with the prescribed drug. He can view the drug details and approve the request for the retail of the drug.

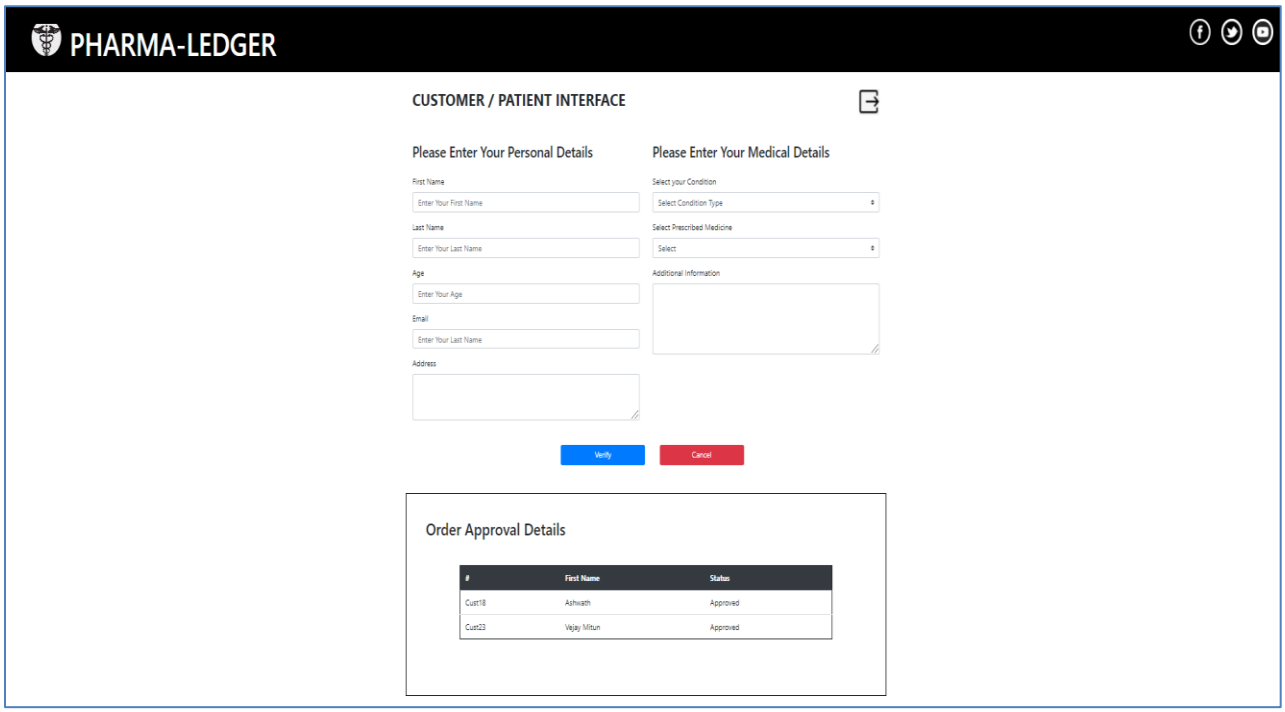






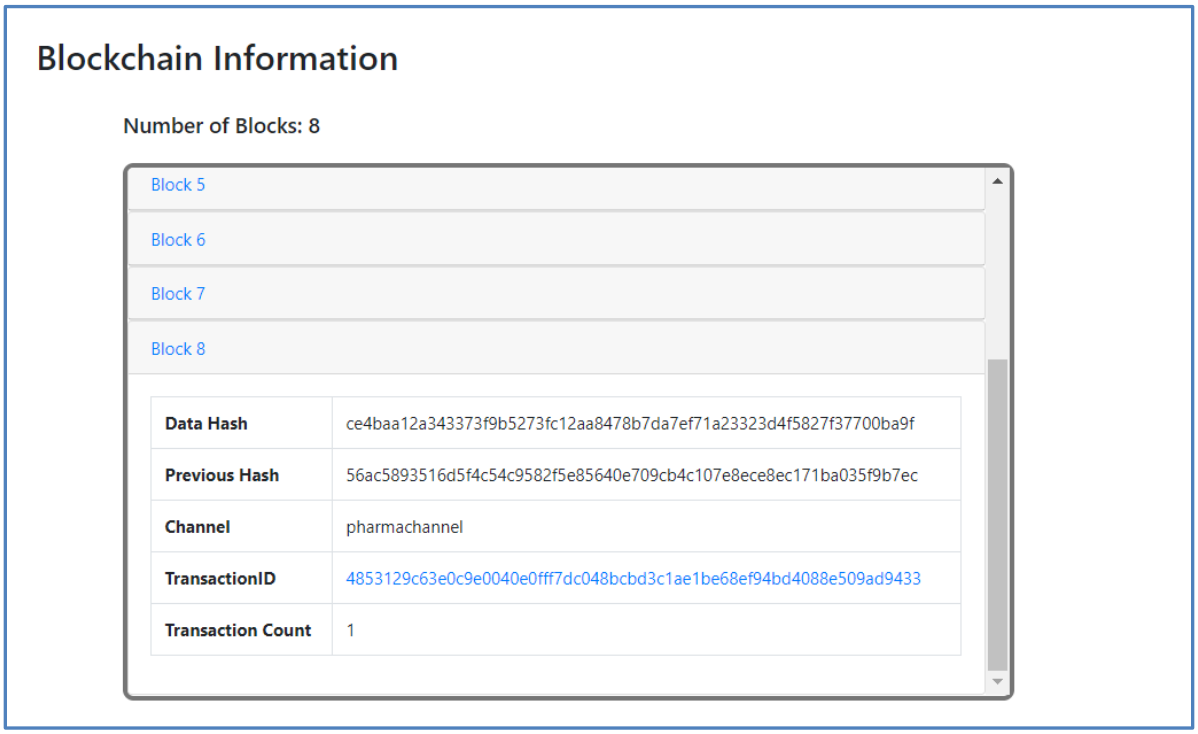
Approved Drug Request for Customer

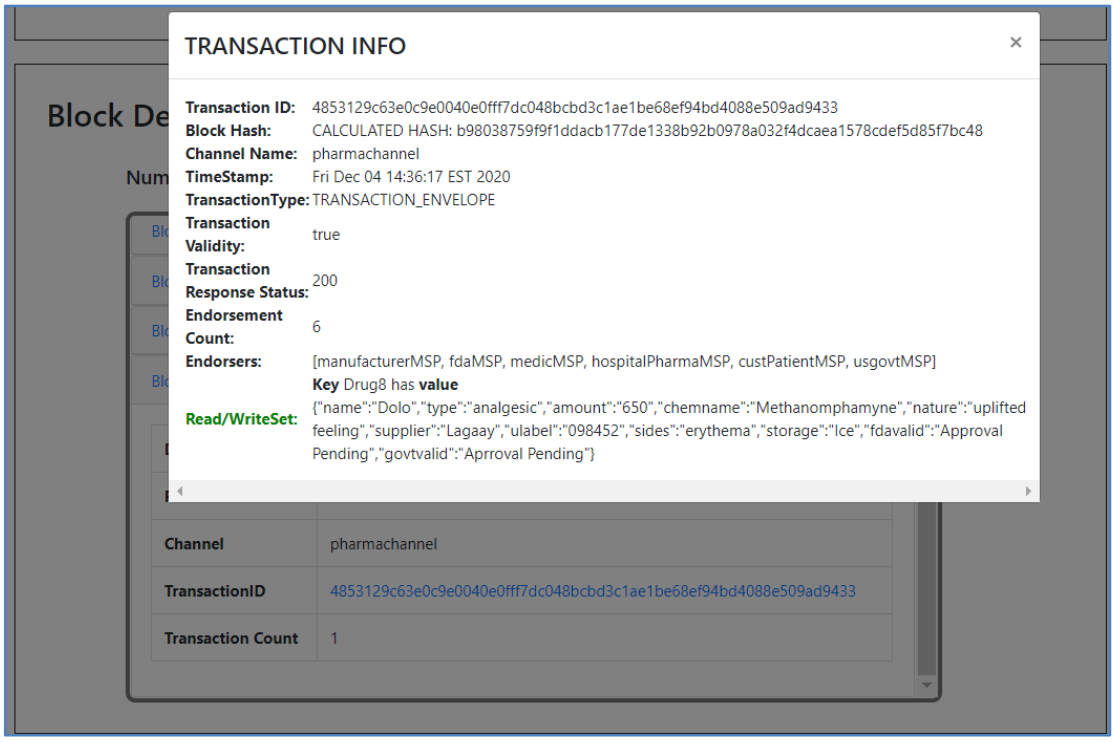
The request for drug retail to the customer which was approved by the doctor, is reflected in this screen and the customer is thus approved to purchase the drug.



Thus, a complete chain of events starting from the manufacturer, supplying the drug, to the FDA and government entity approving the drugs, finally to the customer purchasing the approved/prescribed drugs was simulated. All drug related activities (approval, addition, and query) were tracked and done through the distributed ledger. Every peer page will have a copy of the blockchain itself (so they can view the ledger and transaction details, for transparency)

Ledger and Block Information in each peer interface





Demo Link/Video:

A comprehensive video demo that shows how the underlying blockchain network will look like, the user interface, and a sample code flow as well,

https://youtu.be/sq-0NH2bb5U

Points to Note:

- Hyperledger Fabric provides samples and binaries, whose configurations were extended to realize this use-case.
- Binaries provided by Fabric (the components of Fabric) – peer, orderer, cryptogen, configtxgen etc. can be found in the bin folder.
- The **configtxgen** tool is used to generate the channel and orderer artifacts. This tool mainly creates the genesis block for the blockchain and gives the permissions for each peer. The configurations are defined in **configtx.yaml.** Generated files can be found in the channel-artifacts folder.
- **cryptogen** as mentioned before is used to generate certificates used to identify and authorize various components in the network. The configurations are defined in **crypto-config.yaml.** Generated files can be found in the crypto-config folder.
- There are templates for other consensus protocols as well (provided by Fabric), which can be used if needed (etcdraft, or RAFT – which is beyond scope for this app)
- Every network configuration related file can be found in the folder drugTrackerNetwork.
- In the client webapp, we have configured the SDK to use certificate files stored in the crypto-config directory (which we put in the USER_HOME folder) in order to access the network. You can find the configuration in **BlockChainHFClient.java** for access to the network
- Validations with respect to drug related fields are done at the client-side itself for ease.
- For Ethereum, there is a tool called Ganache to inspect the states of the blockchain. In our application, we show the blockchain itself in every peer interface and show the details in each block as well (Information retrieved through the Fabric Java SDK)
- Every time a transaction is made (like a create drug, approve drug etc.), you get an alert with a transactionID. You can use it to verify block details in the blockchain.
- To stop the network and remove the container in the drugTrackerNetwork folder run the command – **sh drugledgerscript.sh -m down**
- Remember that whenever you bring up the network, you need to copy the crypto-config folder to your HOME directory, to access it from the webapp.

Conclusion:

Through this guide, we saw what is special about Hyperledger Fabric, and how it can be incorporated for a drug tracking supply chain module. Flow diagrams, screenshots, code snippets and demo links were given to show this could act as forefront for a full-fledged scenario. When it comes to enterprise-grade applications, Hyperledger Fabric comes on top, owing to its permissioned and easy plug and play capabilities.

This is a simulation of how a blockchain environment can be used for tracking the movement of drugs. As you can, this seems like a good-enough module that can be developed to a full-fledged scenario. Fabric is said to be faster than other private blockchain frameworks like Corda etc. and can process up to 1000 transactions per second. Since it is permissioned, only known participants can be part of the network, hence a smaller number of nodes and hence maximum efficiency towards computing data. Existing drug tracking system involves a lot of paperwork and manual labor that takes time, and this can be easily avoided through this system. There is also the scope for using IOT based devices attached to the drugs (for example when transporting from one place to another) that emit events to the blockchain ledger for better tracking.

References

- Hyperledger fabric: a distributed operating system for permissioned blockchains, Androulaki et al., Proceedings of the Thirteenth EuroSys Conference. ACM, 2018. [PDF]
- Blockchains and smart contracts for the internet of things., Christidis et al, Ieee Access 4 (2016) [PDF]
- Blockchain Technology in Pharmaceutical Industry to Prevent Counterfeit Drugs March 2018, International Journal of Computer Applications 180(25):8-12 [URL]
- Blockchain Implementations and Use Cases for Supply Chains – A Survey, January 2020 IEEE Access PP(99):1-1 [URL]
- 5 best private blockchain frameworks [URL]
- LedgerDomain Case Study - [URL]
- Hyperledger Fabric Samples/Binaries - [URL]
- Hyperledger Fabric Smart Contracts and Chaincode [URL]
- Hyperledger Fabric Java SDK Doc – [URL]