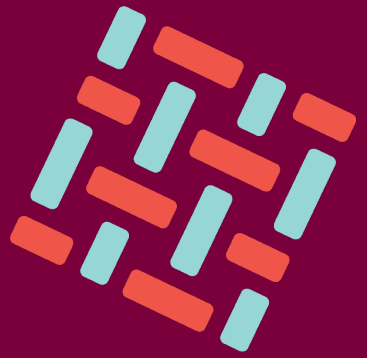
Vilnius University

Blockchain Supply Chain with Hyperledger Fabric



HYPERLEDGER FABRIC

Prof. dr. Remigijus Paulavičius & dr. Ernestas Filatovas, 2022

Outline

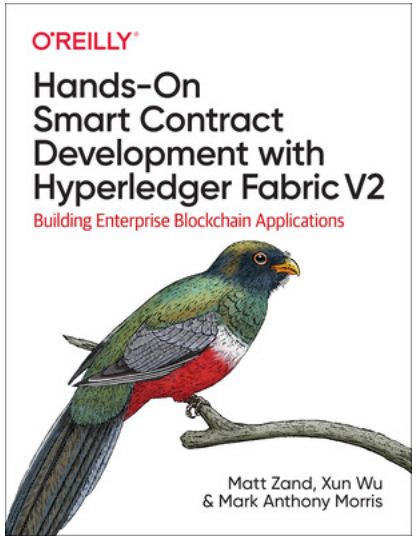
1. Designing a Blockchain Supply Chain
2. Writing Chaincode as a Smart Contract
3. Compiling and Deploying Fabric Chaincode
4. Running and Testing the Smart Contract
5. Developing an Application with Hyperledger Fabric through the SDK

Goals of this lecture

- In this lecture, we will design and build a **simple supply chain blockchain** application called **Pharma Ledger Network (PLN)**
  - This project will give you a taste of how **blockchain enables** global business transactions with **greater transparency**, **streamlined supplier onboarding**, better response to disruptions, and a secure environment.
  - Specifically, the PLN project illustrates how blockchain can help manufacturers, wholesalers, and other supply chain members like pharmacies deliver medical supplies.

This lecture will help you achieve the following practical goals:

- Designing a blockchain supply chain
- Writing chaincode as a smart contract
- Compiling and deploying Fabric chaincode
- Running and testing the smart contract
- Developing an application with Hyperledger Fabric through the SDK



Designing a Blockchain Supply Chain

Source code for this lecture:

https://myhsts.org/hyperledger-fabric-book/ch7-hyperledger-fabric-supply-chain-dapp.zip

Vilnius University

Current Issues

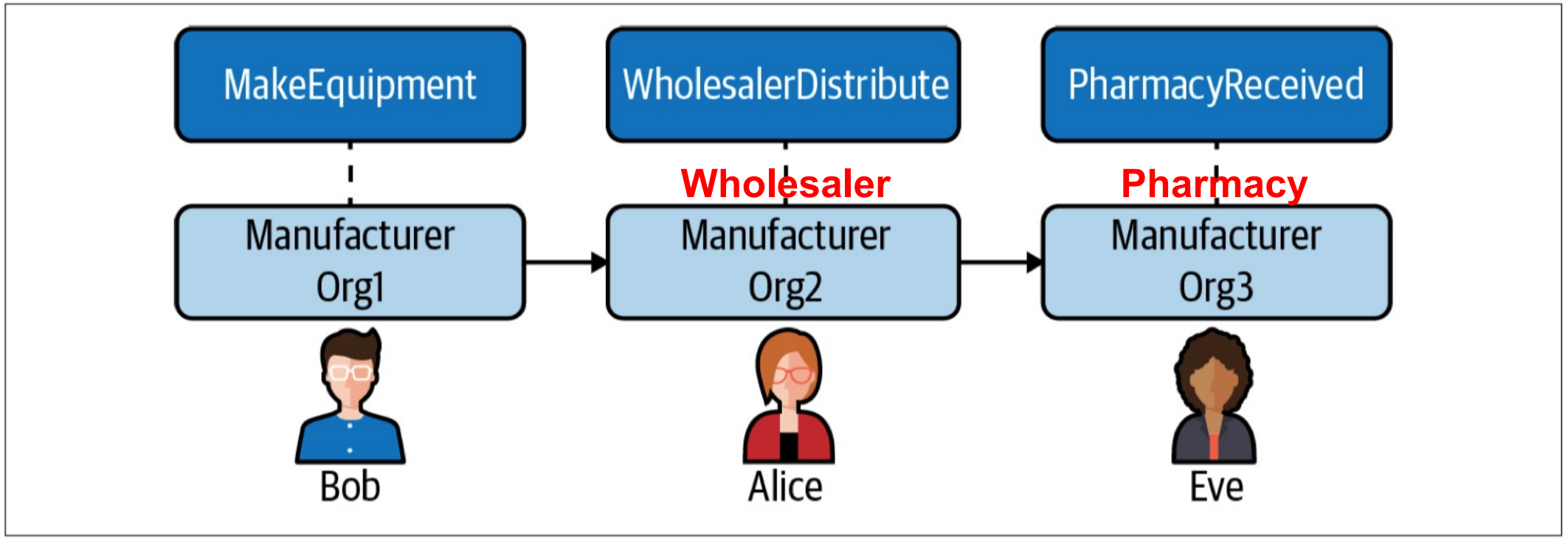
- The **traditional supply chain** usually **lacks transparency** and reliable reporting.
- Large organizations **have built their own systems** to enable global control of their daily operations while recording real-time transactions between suppliers and distributors.
- However, many small companies lack that information and have limited visibility to trace their products at any given moment.
- That means **the transparency from upstream to downstream is very limited** in their entire supply chain product process flow (from production to consumption).
- This could lead to inaccurate reports and a lack of interoperability.

Motivation to use Blockchain

- By design, the **blockchain is a shared-ledger, transparent, immutable, and secure decentralized system**.
- It is considered a good solution for traditional supply chain industries at registering, controlling, and transferring assets.
- A **smart contract**, which **defines a business function**, can be **deployed in blockchain** and then **accessed by multiple parties** in the blockchain network.
- **Each member in the blockchain** will be assigned **unique identifiers to sign and verify the blocks they add** to the blockchain.
- During the life cycle of the supply chain, when authorized members in a consortium network invoke a smart contract function, the state data will be updated, after which current assets' status and the transaction data will become a permanent record in the ledger.
- Likewise, the processes related to assets can be easily and quickly moved from one step to another.
- The **digital transactions in the ledger** can be **tracked, shared, and queried by all supply chain participants in real time**.
- It provides organizations with new opportunities to correct problems within their supply chain system as it revolves around a single source of truth.

Understanding the Supply Chain Workflow (1)

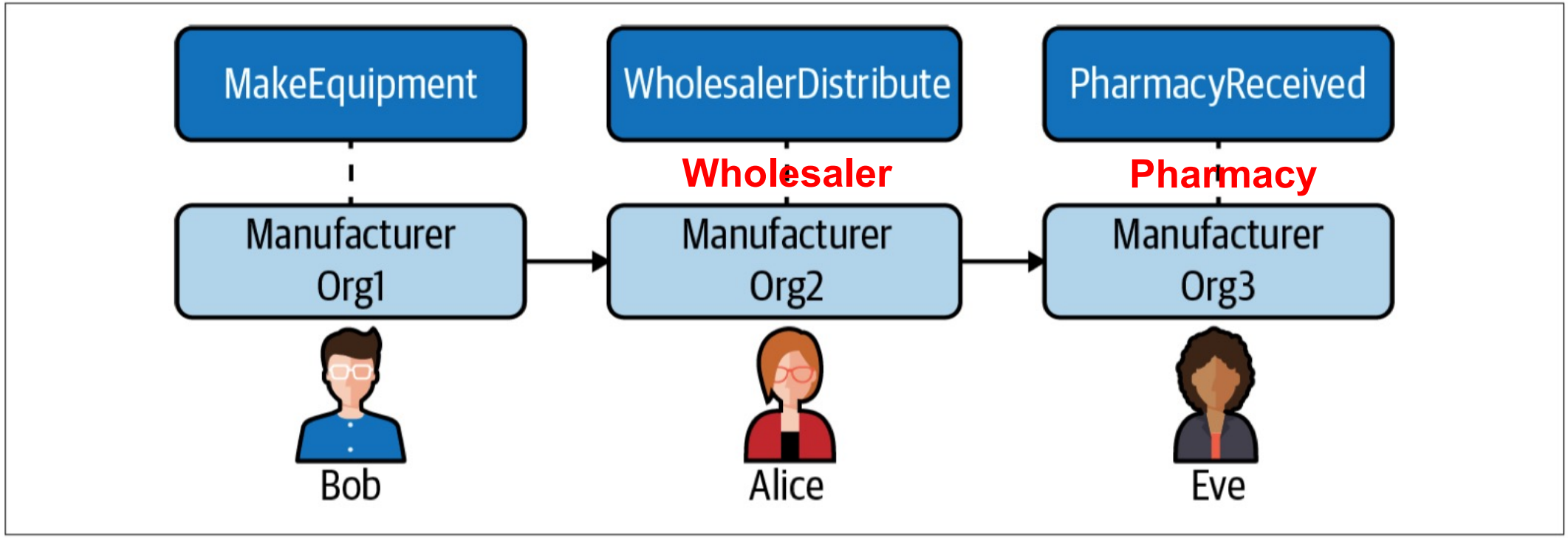
- Let's look at organizations in the PLN business scenario, as shown in Fig.
- For demonstration purposes, we simplified the pharma ledger process, as it can be much more complex in the real world



Organizations in the PLN

Understanding the Supply Chain Workflow (2)

- Our PLN process is divided into the following three steps:
  1. A **manufacturer makes equipment** and **ships it to the wholesaler**.
  2. A **wholesaler distributes** the **equipment to** the **pharmacy**.
  3. The **pharmacy, as a consumer**, **receives the equipment**, and the supply chain workflow is completed.



Organizations in the PLN

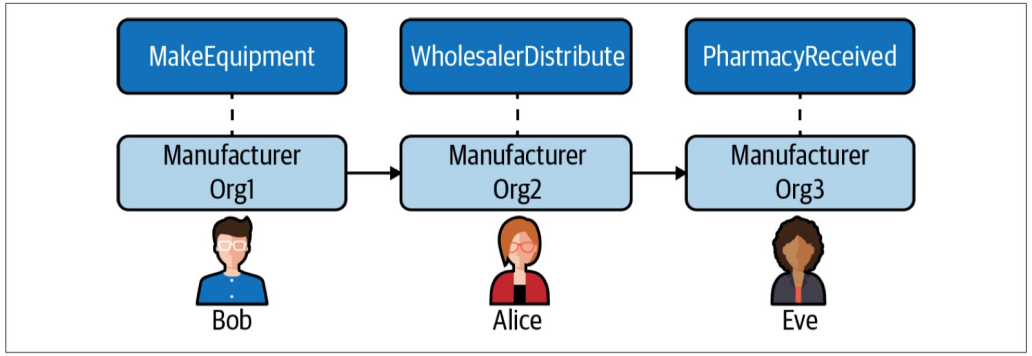
Defining a Consortium (1)

- As we can see from the process workflow, our PLN involves three organizations:
  - **1) manufacturer, 2) wholesaler,** and **3) pharmacy.**
- These **three entities will join in building a consortium network** to carry out the supply chain business.
- The **consortium members can create users, invoke smart contracts, and query blockchain data**.
- The table depicts the organizations and users in the PLN consortium.

Organization name	User	MSP	Peer
Manufacturer	Bob	Org1MSP	*peer0.org1.example.com*
Wholesaler	Alice	Org2MSP	*peer0.org2.example.com*
Pharmacy	Eve	Org3MSP	*peer0.org3.example.com*

Defining a Consortium (2)

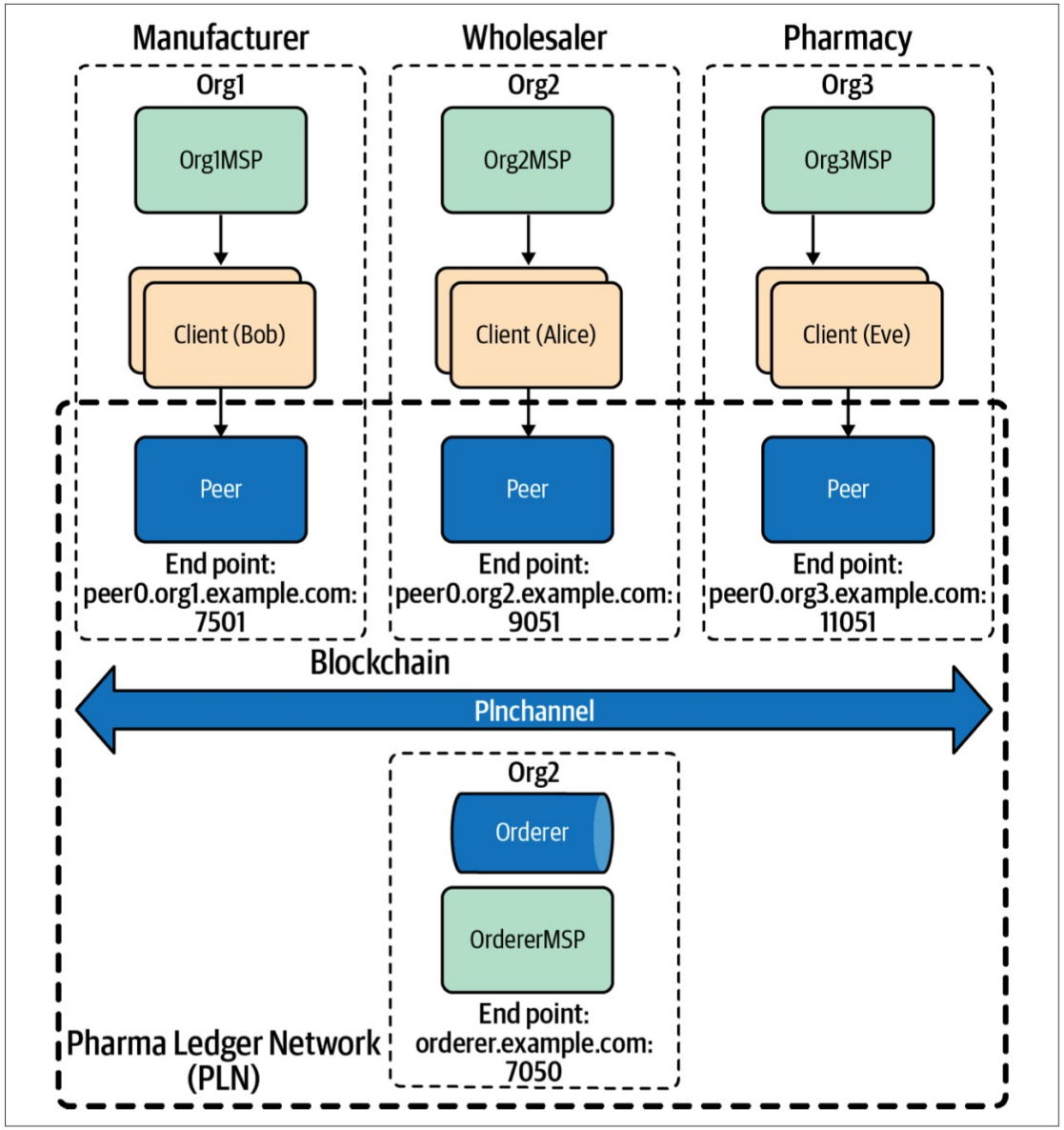
- In our PLN consortium, **each of the 3 organizations** has a **user**, an **MSP**, and a **peer**.
- For the **manufacturer organization**, we **have a user called Bob** as an application user.
  - Org1MSP is an MSP ID to load the MSP definition.
- We define **AnchorPeers** with the hostname peer0.org1.example.com to gossip communication across.
- Similarly, the **wholesaler is the second organization**, **Alice is** its **application user**, and its **MSP ID is Org2MSP**.
- Finally, **Eve is the pharmacy organization user with Org3MSP**.



Organization name	User	MSP	Peer
Manufacturer	Bob	Org1MSP	*peer0.org1.example.com*
Wholesaler	Alice	Org2MSP	*peer0.org2.example.com*
Pharmacy	Eve	Org3MSP	*peer0.org3.example.com*

Hyperledger Fabric network topology

- Since **installing and deploying PLN in multiple physical nodes may not be within the scope** of this lecture, **we define one peer with four organizations, representing** the **manufacturer, wholesaler, pharmacy,** and **orderer nodes**.
- The channel **plnchannel provides** a **private communications** mechanism **used by the orderer and the other three organizations to execute and validate the transactions**.



Reviewing the PLN Life Cycle

- As we mentioned in the previous section, the PLN life cycle has three steps:
  - the manufacturer makes equipment and ships to the wholesaler;
  - the wholesaler distributes the equipment to the pharmacy;
  - and finally, the pharmacy receives the equipment.
- **The entire process can be traced by equipment ID.**
- A piece of equipment with equipment **ID 2000.001** was made by a manufacturer on January 1, with equipment and other attributes and values, as shown in Figure

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: MANUFACTURER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 1, 2021, 10:01:02

Equipment attributes and values

- Each **equipment item is owned by an equipment owner at a certain period of time.**
- In our case, **we define three owner types: manufacturer, wholesaler, and pharmacy**.
- When a manufacturer makes a piece of equipment and records it in the PLN, the **transaction result shows the equipment with a unique identification number of 2000.001 in the ledger**.
- The current owner is **GlobalEquipmentCorp**.
- The **current owner type** and **previous one** are the same — **manufacturer**.
- The **lastUpdated** entry is the date when the transaction was recorded in the PLN

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: MANUFACTURER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 1, 2021, 10:01:02

Equipment state changes for the wholesaler

- After a few weeks, the **manufacturer ships the equipment to the wholesaler**, and the **equipment state will change**, **including ownership, previous and current owner type, and last update**.

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: MANUFACTURER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 1, 2021, 10:01:02

- Let's take a look at which **equipment states change**, as shown in Figure →
  - The **equipment is now owned by GlobalWholesalerCorp**.
  - The **previous owner** type **is** the **manufacturer**.
  - The **last updated date** has **also changed**.

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: WHOLESALER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 20, 2021, 07:12:12

Equipment in the hand of the pharmacy

- After one month, the **pharmacy finally receives this equipment order**.
- The **ownership is now transferred from** the **wholesaler to** the **pharmacy**, as shown in Figure.
- The supply chain flow can be considered closed.
- With the same equipment identity, the peer organization can trace the equipment's entire history of transaction records by looking up the equipment number.
- As you've seen, the entire life cycle has three steps. Originating from the manufacturer, the equipment moves from wholesaler to pharmacy.
  - As such, as a result of making a piece of equipment, the wholesaler distributes and the pharmacy receives the transaction.
  - With all of this design and analysis, we can now start to write our PLN smart contract.

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: MANUFACTURER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 1, 2021, 10:01:02

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: WHOLESALER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 20, 2021, 07:12:12

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: PharmacyCorp
previousOwnerType: WHOLESALER,
currentOwnerType: PHARMACY,
createDateTime: Jan 1, 2021,
lastUpdated: Feb 25, 2021, 11:01:08

Equipment in the hand of the pharmacy

- After one month, the **pharmacy finally receives this equipment order**.
- The **ownership is now transferred from** the **wholesaler to** the **pharmacy**, as shown in Figure.
- The supply chain flow can be considered closed.
- With the same equipment identity, the peer organization can trace the equipment's entire history of transaction records by looking up the equipment number.
- As you've seen, the entire life cycle has three steps. Originating from the manufacturer, the equipment moves from wholesaler to pharmacy.
  - As such, as a result of making a piece of equipment, the **wholesaler distributes** and the **pharmacy receives the transaction**.
  - **With all of this design and analysis, we can now start to write our PLN smart contract**.

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: MANUFACTURER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 1, 2021, 10:01:02

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: GlobalEquipmentCorp
previousOwnerType: MANUFACTURER,
currentOwnerType: WHOLESALER,
createDateTime: Jan 1, 2021,
lastUpdated: Jan 20, 2021, 07:12:12

equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: PharmacyCorp
previousOwnerType: WHOLESALER,
currentOwnerType: PHARMACY,
createDateTime: Jan 1, 2021,
lastUpdated: Feb 25, 2021, 11:01:08

Writing Chaincode as a Smart Contract

Vilnius University

Writing Chaincode as a Smart Contract

- In Hyperledger Fabric, a **smart contract is a program that implements the business logic** and **manages the world state of a business object during its life cycle**.
- **During deployment**, this **contract will be packaged into** the **chaincode** and **installed on** each **endorsing peer** that runs in a secured Docker container.
- The Hyperledger Fabric smart contract can be programmed in Go, **JavaScript**, Java, and Python.

Project Structure

- To start our PLN smart contract development, **first we need to create our smart contract project**.
- Since **we have three organizations**, **all peers must agree and approve of the new version of the smart contract** that will be installed and deployed to the network.
- For our PLN, we will assume they are all the same
- The project structure is shown here
- The **package.json** file defines the two most important fabric libraries:

```
19      "dependencies": {
20          "fabric-contract-api": "^2.1.2",
21          "fabric-shim": "^2.1.2"
```

```
manufacturer
├── application
│   ├── app.js
│   ├── package.json
│   ├── public
│   │   ├── css
│   │   │   └── style.css
│   │   ├── images
│   │   │   └── wait.gif
│   │   └── js
│   │       └── plnClient.js
│   ├── services
│   │   ├── equipmentService.js
│   │   ├── queryHistoryService.js
│   │   ├── queryService.js
│   │   └── walletsService.js
│   └── views
│       └── index.ejs
└── contract
    ├── index.js
    ├── lib
    │   └── pharmaledgercontract.js
    └── package.json
```

Smart Contract Class (1)

- We define a **smart contract** called **pharmaledgercontract.js**.
- **fabric-contract-api** provides the **Contract** interface. It has two critical classes that every smart contract needs to use, **Contract** and **Context**:

```
20    // Fabric smart contract classes
21    const { Contract, Context } = require('fabric-contract-api');
```

- **Contract** has beforeTransaction, afterTransaction, unknownTransaction, and createContext methods that are optional and overridable in the subclass.
- The **Context** class provides the transactional context for every transactional invocation.
  - It can be overridden for additional application behavior to support smart contract execution.

Smart Contract Class (2)
Constructor and unique namespace

- Let's first define **PharmaLedgerContract** with a constructor: **org.pln.PharmaLedgerContract** gives a very descriptive name **with a unique namespace** for our contract.
- The **unique contract namespace is important to avoid conflict** when a shared system has many contracts from different users and operations:

```
23  /**
24   * Define PharmaLedger smart contract by extending Fabric Contract class
25   *
26   */
27  class PharmaLedgerContract extends Contract {
28
29      constructor() {
30          // Unique namespace pcn - PharmaChainNetwork when multiple contracts per chaincode file
31          super('org.pln.PharmaLedgerContract');
32      }
```

Smart Contract Class (3)
Transaction Logic

- As we discussed, **PharmaLedgerContract** will need **three business functions** to **move** the equipment **owner from** the **manufacturer to** the **wholesaler**, and finally **pharmacy**:

```
async makeEquipment(ctx, manufacturer, equipmentNumber, equipmentName, ownerName) {
    // makeEquipment logic
}
async wholesalerDistribute(ctx, equipmentNumber, ownerName) {
    // wholesalerDistribute logic
}
async pharmacyReceived(ctx, equipmentNumber, ownerName) {
    // pharmacyReceived logic
}
```

- The **manufacturer will be initialized**, and an equipment entry is created.
- As you will notice, **these functions accept a context (ctx)** as the default **first parameter with equipment related arguments** (manufacturer, equipmentNumber, equipmentName, ownerName) from client input.

Smart Contract Class (4)
makeEquipment (1)

- When **makeEquipment** is called, the function expects **four equipment attributes** from the client and assigns it to new equipment:
- But first, the function uses the JavaScript **new Date** to **get the current date time** and assign it to the **createDateTime** and **lastUpdated** date time.

```
42      /**
43       * Create pharma equipment
44       *
45       * @param {Context} ctx the transaction context
46       * @param {String} equipment manufacturer
47       * @param {String} equipmentNumber for this equipment
48       * @param {String} equipment name
49       * @param {String} name of the equipment owner
50       */
51      async makeEquipment(ctx, manufacturer, equipmentNumber, equipmentName, ownerName) {
52          console.info('============= START : makeEquipment call ===========');
53          let dt = new Date().toString();
54          const equipment = {
55              equipmentNumber,
56              manufacturer,
57              equipmentName,
58              ownerName,
59              previousOwnerType: 'MANUFACTURER',
60              currentOwnerType: 'MANUFACTURER',
61              createDateTime: dt,
62              lastUpdated: dt
63          };
64          await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(equipment)));
65          console.info('============= END : Create equipment ===========');
66      }
```

Smart Contract Class (5)
makeEquipment (2)

- At the end of makeEquipment, **ctx.stub.putState** will **store the equipment's initial state value** with the equipment number key **on the ledger**.
- The **equipment JSON data will be stringified** using **JSON.stringify**, **then converted to a buffer**.
  - The buffer conversion is required by the shim API to communicate with the peer.
- When transaction data is submitted, each peer will validate and commit a transaction.

```
42      /**
43       * Create pharma equipment
44       *
45       * @param {Context} ctx the transaction context
46       * @param {String} equipment manufacturer
47       * @param {String} equipmentNumber for this equipment
48       * @param {String} equipment name
49       * @param {String} name of the equipment owner
50       */
51      async makeEquipment(ctx, manufacturer, equipmentNumber, equipmentName, ownerName) {
52          console.info('============= START : makeEquipment call ===========');
53          let dt = new Date().toString();
54          const equipment = {
55              equipmentNumber,
56              manufacturer,
57              equipmentName,
58              ownerName,
59              previousOwnerType: 'MANUFACTURER',
60              currentOwnerType: 'MANUFACTURER',
61              createDateTime: dt,
62              lastUpdated: dt
63          };
64          await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(equipment)));
65          console.info('============= END : Create equipment ===========');
66      }
```

Smart Contract Class (6)
wholesalerDistribute (1)

- **After** the equipment **record is created by** the **manufacturer, the wholesaler** and **pharmacy will** just **need** to **update ownership** to track the current owner. Both functions are similar.
- We query current equipment ledger data by calling ctx.stub.getState(equipmentNumber)
- Once data returns, we need to make sure equipmentAsBytes is **not empty** and equipmentNumber **is a valid number**.
- Since ledger data is in JSON string byte format, that data needs to convert encoded data to a readable JSON format by using Buffer.from().toString('utf8')

```
    /**
     * Manufacturer send equipment To Wholesaler
     *
     * @param {Context} ctx the transaction context
     * @param {String} equipmentNumber for this equipment
     * @param {String} name of the equipment owner
    */
    async wholesalerDistribute(ctx, equipmentNumber, ownerName) {
        console.info('============= START : wolesalerDistribute call ===========');
        const equipmentAsBytes = await ctx.stub.getState(equipmentNumber);
        if (!equipmentAsBytes || equipmentAsBytes.length === 0) {
            throw new Error(`${equipmentNumber} does not exist`);
        }
        let dt = new Date().toString();
        const strValue = Buffer.from(equipmentAsBytes).toString('utf8');
        let record;
        try {
            record = JSON.parse(strValue);
            if(record.currentOwnerType!=='MANUFACTURER') {
               throw new Error(` equipment - ${equipmentNumber} owner must be MANUFACTURER`);
            }
            record.previousOwnerType= record.currentOwnerType;
            record.currentOwnerType = 'WHOLESALER';
            record.ownerName = ownerName;
            record.lastUpdated = dt;
        } catch (err) {
            console.log(err);
            throw new Error(`equipmet ${equipmentNumber} data can't be processed`);
        }
        await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(record)));
        console.info('============= END : wolesalerDistribute  ===========');
    }
```

Smart Contract Class (7)
wholesalerDistribute (2)

- We then **verify that the current equipment owner type is the manufacturer** by using the returned data and **update the current owner to wholesaler**.
- Once all these conditions are met, **ctx.stub.putState** is called again.
- The equipment owner state would be **updated to the wholesaler with the current timestamp**.
- But as an immutable transaction log, all historical changes of the world state **will permanently store in the ledger**.

```
/**
 * Manufacturer send equipment To Wholesaler
 *
 * @param {Context} ctx the transaction context
 * @param {String} equipmentNumber for this equipment
 * @param {String} name of the equipment owner
*/
async wholesalerDistribute(ctx, equipmentNumber, ownerName) {
    console.info('============= START : wolesalerDistribute call ===========');
    const equipmentAsBytes = await ctx.stub.getState(equipmentNumber);
    if (!equipmentAsBytes || equipmentAsBytes.length === 0) {
        throw new Error(`${equipmentNumber} does not exist`);
    }
    let dt = new Date().toString();
    const strValue = Buffer.from(equipmentAsBytes).toString('utf8');
    let record;
    try {
        record = JSON.parse(strValue);
        if(record.currentOwnerType!=='MANUFACTURER') {
           throw new Error(` equipment - ${equipmentNumber} owner must be MANUFACTURER`);
        }
        record.previousOwnerType= record.currentOwnerType;
        record.currentOwnerType = 'WHOLESALER';
        record.ownerName = ownerName;
        record.lastUpdated = dt;
    } catch (err) {
        console.log(err);
        throw new Error(`equipmet ${equipmentNumber} data can't be processed`);
    }
    await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(record)));
    console.info('============= END : wolesalerDistribute  ===========');
}
```

Smart Contract Class (8)
pharmacyReceived

- The pharmacyReceived function is similar to wholesalerDistribute.
- It **needs to validate** that **the current owner is** the **wholesaler** and then **transfer ownership to the pharmacy** before updating the equipment record.
- After we implement all three equipment business functions, the ledger still needs:
  - a **query function to search current equipment data**,
  - and a **query history function to get all of the historical records**.

```
    /**
     * Wholesaler send equipment To Pharmacy
     *
     * @param {Context} ctx the transaction context
     * @param {String} equipmentNumber for this equipment
     * @param {String} name of the equipment owner
    */
    async pharmacyReceived(ctx, equipmentNumber, ownerName) {
        console.info('============= START : pharmacyReceived call ===========');
        const equipmentAsBytes = await ctx.stub.getState(equipmentNumber);
        if (!equipmentAsBytes || equipmentAsBytes.length === 0) {
            throw new Error(`${equipmentNumber} does not exist`);
        }
        let dt = new Date().toString();
        const strValue = Buffer.from(equipmentAsBytes).toString('utf8');
        let record;
        try {
            record = JSON.parse(strValue);
            //make sure owner is wholesaler
            if(record.currentOwnerType!=='WHOLESALER') {
               throw new Error(` equipment - ${equipmentNumber} owner must be WHOLESALER`);
            }
            record.previousOwnerType= record.currentOwnerType;
            record.currentOwnerType = 'PHARMACY';
            record.ownerName = ownerName;
            record.lastUpdated = dt;
        } catch (err) {
            console.log(err);
            throw new Error(`equipmet ${equipmentNumber} data can't be processed`);
        }
        await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(record)));
        console.info('============= END : pharmacyReceived  ===========');
    }
```

Smart Contract Class (9)
queryByKey

- **ChaincodeStub** is implemented by the **fabric-shim library** and provides GetState and GetHistoryForKey functions.
- In our case, the query definition is straightforward: we just **need to call ctx.stub.getState** to get the **corresponding result**.

```
132     /**
133      * query ledger record By Key
134      *
135      * @param {Context} ctx the transaction context
136      * @param {String} key for record
137     */
138     async queryByKey(ctx, key) {
139         let value = await ctx.stub.getState(key);
140         const strValue = Buffer.from(value).toString('utf8');
141         let record;
142             try {
143                 record = JSON.parse(strValue);
144             } catch (err) {
145                 console.log(err);
146                 record = strValue;
147             }
148         return JSON.stringify({
149            Key: key, Record: record
150         });
151     }
```

Smart Contract Class (10)
GetHistoryForKey

- GetHistoryForKey **returns all historical transaction key values across time**.
- We can iterate through these records and convert them to a JSON byte array and send the data back as a response.
- The timestamp tells us when the equipment state was updated.
- **Each record contains a related transaction ID and a timestamp**
- That is all for the smart contract function we will implement for our PLN.
- Next, we will compile and deploy the Fabric chaincode.

```
152     async queryHistoryByKey(ctx, key) {
153         console.info('getting history for key: ' + key);
154         let iterator = await ctx.stub.getHistoryForKey(key);
155         let result = [];
156         let res = await iterator.next();
157         while (!res.done) {
158           if (res.value) {
159             const obj = JSON.parse(res.value.value.toString('utf8'));
160             result.push(obj);
161           }
162           res = await iterator.next();
163         }
164         await iterator.close();
165         console.info(result);
166         return JSON.stringify(result);
167     }
168 }
169 module.exports = PharmaLedgerContract;
```

Compiling and Deploying Fabric Chaincode

Vilnius University

Compiling and Deploying Fabric Chaincode

- Before deploying our contract, we need to set up the Fabric network.
- Before advancing any further, we first need to meet some prerequisites (the same as in Lecture 13) + **Go version 1.14+**
- To set up a network, we
  - **generate crypto material for an organization by using Cryptogen,**
  - **create a consortium,**
  - **and then bring up PLN with Docker Compose.**
- Let's first set up our project.

Review the Project Structure

- We have defined all setup scripts and configuration files for our PLN project.
- The project structure is organized as follows:
- Let’s take a look at important configurations.

```
├── loadFabric.sh
└── pharma-ledger-network
    ├── configtx
    │   └── configtx.yaml
    ├── docker
    │   ├── docker-compose-ca.yaml
    │   └── docker-compose-pln-net.yaml
    ├── net-pln.sh
    ├── organizations
    │   ├── ccp-generate.sh
    │   ├── ccp-template.json
    │   ├── ccp-template.yaml
    │   ├── cryptogen
    │   ├── manufacturer
    │   ├── pharmacy
    │   └── wholesaler
    ├── scripts
    │   ├── createChannel.sh
    │   ├── deploySmartContract.sh
    │   ├── invokeContract.sh
    │   ├── monitor.sh
    │   └── utils.sh
    └── system-genesis-block
```

Cryptogen

- Four crypto configurations are in the **cryptogen** folder **for the orderer and the other three peer organizations**.
- **OrdererOrgs** defines ordering nodes and creates an organization definition.
- **PeerOrgs** defines peers, organization, and managing peer nodes.
- As we know, **requires a CA**. running components in the network
- The **Fabric Cryptogen tool** will **use** those **four crypto configuration** files **to generate** the required **X.509 certificates** for all organizations.

```
cryptogen
├── crypto-config-orderer.yaml
├── crypto-config-org1.yaml
├── crypto-config-org2.yaml
└── crypto-config-org3.yaml
```

OrdererOrgs and PeerOrgs

- For **OrdererOrgs**, we define the following crypto configuration

```
$ cat crypto-config-orderer.yaml
# O'Reilly - Accelerated Hands-on Smart Contract Development with Hyperledger Fabric V2
# farma ledger supply chain network
# Author: Brian Wu
#
# ---------------------------------------------------------------------------
# "OrdererOrgs" -define ordering node and Create an organization definition
# ---------------------------------------------------------------------------
OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
    Domain: example.com
    EnableNodeOUs: true
    # ---------------------------------------------------------------------------
    # "Specs" - See PeerOrgs for complete description
    # SANS: (Optional) Specifies one or more Subject Alternative Names
    # to be set in the resulting x509.
    # ---------------------------------------------------------------------------
    Specs:
      - Hostname: orderer
        SANS:
          - localhost
```

- For **PeerOrgs**, we define the following crypto configuration for **Org1 (manufacturer)**. The other two orgs are similar:

```
$ cat crypto-config-org1.yaml
# O'Reilly - Accelerated Hands-on Smart Contract Development with Hyperledger Fabric
# farma ledger supply chain network
# Author: Brian Wu
#

# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# This file define peer org1 manufacturer Definition
# ---------------------------------------------------------------------------
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org1
  # EnableNodeOUs:  enables the identify classification
  # ---------------------------------------------------------------------------
  - Name: Org1
    Domain: org1.example.com
    EnableNodeOUs: true
    # ---------------------------------------------------------------------------
    # "Template"
    # ---------------------------------------------------------------------------
    # Allows for the definition of 1 or more hosts that are created sequentially
    # from a template.
    # ---------------------------------------------------------------------------
    Template:
      Count: 1
      SANS:
        - localhost
    # ---------------------------------------------------------------------------
    # "Users"
    # ---------------------------------------------------------------------------
    # Count: The number of user accounts _in addition_ to Admin
    # ---------------------------------------------------------------------------
    Users:
      Count: 1
```

We set **EnableNodeOUs** to true, which enables the identity classification.

Configtx (in configtx/ folder)

- The **configtx.yaml** file will generate **OrdererSystemChannelGenesis** and related artifacts by **configtx.yaml** configuration.
- In the **configtx.yaml** Organizations section:
  - We define **OrdererOrg** and the **other three peer organizations—Org1, Org2, and Org3**, representing **manufacturer, wholesaler, and pharmacy, respectively**.
  - Each organization will define its Name, ID, MSPDir, and **AnchorPeers**.

    MSPDir describes Cryptogengenerated output MSP directories.
  - **AnchorPeers specifies the peer node's host and port**.

    It updates transactions based on peer policy for communication between network organizations and finds all active participants of the channel:
  - The **Organization Policies section** defines who needs to approve the organization resource.
  - The **Profiles section defines** how to generate PharmaLedgerOrdererGenesis, including order configuration and organizations in the PLN consortiums

Install Binaries and Docker Images

- We have reviewed important configurations in order to run the PLN network.
- The **net-pln.sh** script will bring up the PLN network, but **we first need to download and install Fabric binaries to your system**.
- Under the root project folder is a file called **loadFabric.sh**;
  - run the following command to load Fabric binaries and configs:

```
$ ./loadFabric.sh
```

- This will install the Hyperledger Fabric platform-specific binaries and config files into the **/bin** and **/config** directories under the project.
- To **check installed Fabric images** run:

```
$ docker images -a
```

Start the PLN Network

Now, let's bring up the PLN network. Open a terminal window and **run** net-pln.sh under the **pharma-ledger-network** folder: **./net-pln.sh up**

```
~/HyperFabric/supply-chain/pharma-ledger-network 10:28:21
$ ./net-pln.sh up                                                                                    ‹system›
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb'

LOCAL_VERSION=2.1.0
DOCKER_IMAGE_VERSION=2.1.0
Creating peer0.org2.example.com ... done
Creating orderer.example.com    ... done
Creating peer0.org3.example.com ... done
Creating peer0.org1.example.com ... done
CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS                  PORTS                                   NAMES
ca5b2a90b12d   hyperledger/fabric-peer:latest      "peer node start"   4 seconds ago   Up Less than a second   7051/tcp, 0.0.0.0:11051->11051/tcp      peer0
.org3.example.com
86242b416487   hyperledger/fabric-peer:latest      "peer node start"   4 seconds ago   Up 1 second             0.0.0.0:7051->7051/tcp                  peer0
.org1.example.com
28808b2b4634   hyperledger/fabric-orderer:latest   "orderer"           4 seconds ago   Up Less than a second   0.0.0.0:7050->7050/tcp                  order
er.example.com
b72539eb4305   hyperledger/fabric-peer:latest      "peer node start"   4 seconds ago   Up 1 second             7051/tcp, 0.0.0.0:9051->9051/tcp        peer0
.org2.example.com
```

- We have **four organizations**, including **three peers** and **one orderer**, that are running in the **net_pln network**.
- In the next step, we **will use the script to create a PLN channel for all orgs.**

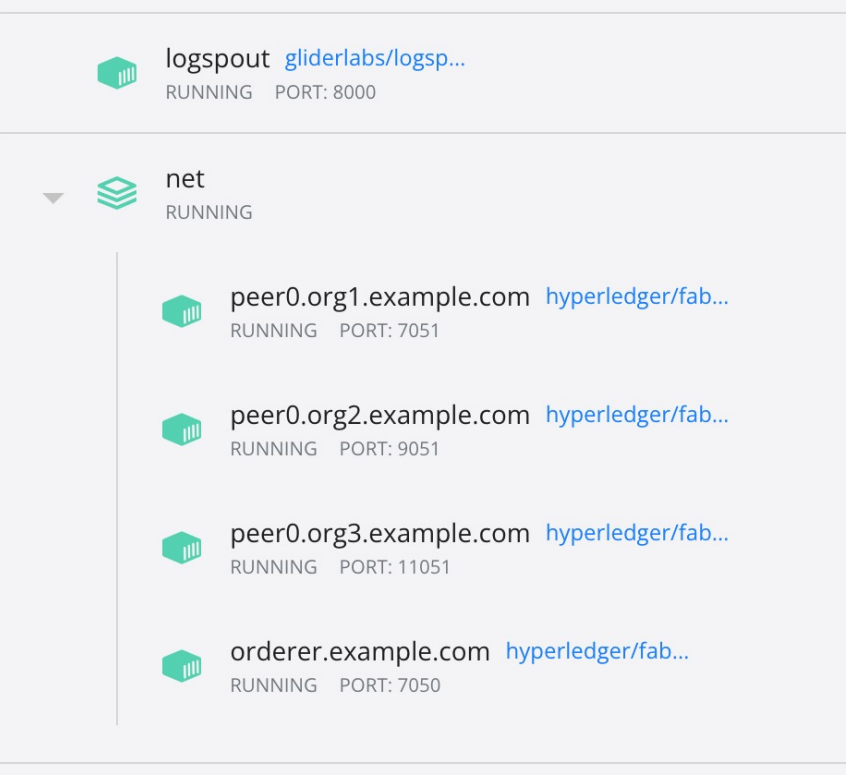
Monitor the PLN Network (1)

- The **Fabric images** in the PLN network **are Docker based**
- During project development or the production life cycle, you may encounter many errors.
- Log monitoring is one of the most important things to do from a DevOps standpoint for troubleshooting the code.
- **Logspout** is an open source container log tool for monitoring Docker logs.
- It collects Docker's logs from all nodes in your cluster to be aggregated into one place.
- **In our PLN project, we will use Logspout to monitor channel creation, smart contract installation, and other actions.**

Monitor the PLN Network (2)

- Navigate to the **pharma-ledger-network** folder, open a new terminal window and run: **./net-pln.sh monitor-up**
- This terminal window will now show the PLN network container output for the remainder of the project development.

```
~/HyperFabric/supply-chain/pharma-ledger-network ⌚ 10:56:43
$ ./net-pln.sh monitor-up
Starting docker log monitoring on network 'net_pln'

Starting monitoring on all containers on the network net_pln
Unable to find image 'gliderlabs/logspout:latest' locally
latest: Pulling from gliderlabs/logspout
8572bc8fb8a3: Pull complete
bd801371a862: Pull complete
58100c398b34: Pull complete
Digest: sha256:2d81c026e11ac67f7887029dbfd7d36ee986d946066b45c1dabd966278eb5681
Status: Downloaded newer image for gliderlabs/logspout:latest
fb4a5a5b90aacbd6ece0cb95c9ac5d02c43aa7cb400dd9557d1ad3f96c46d809
```



Create a PLN Channel (1)

- To **create the channel**, we will use the **configtxgen** CLI tool **to generate a genesis block**, and then we'll use peer channel commands to join a channel with other peers.
- Creating a PLN channel requires several steps:
  1. Generate a channel configuration transaction file.
  2. Create an AnchorPeer configuration transaction file
  3. Create a channel by using the peer channel command
  4. Join all peers into this channel
  5. Select at least one peer as an anchor peer.

All this script logic can be found in **scripts/createChannel.sh**

Create a PLN Channel (2)

- We use **net-pln.sh to perform all these five steps** by running the following command: ./net-pln.sh createChannel
- Once channel creation is completed, you should see the following log:

```
***** [Step: 5]: start call updateAnchorPeers 3 on peer: peer0.org3, channelID: plnchannel, smartcontract: , version , sequence  *****
Using organization 3
2021-12-16 15:09:07.620 EET [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-12-16 15:09:07.800 EET [channelCmd] update -> INFO 002 Successfully submitted channel update
***** completed call updateAnchorPeers, updated peer0.org3 on anchorPeers on channelID: plnchannel, smartcontract: , version , sequence  *****

***** completed call updateOrgsOnAnchorPeers, anchorPeers updated on channelID: plnchannel, smartcontract: , version , sequence  *****

========== Pharma Ledger Network (PLN) Channel plnchannel successfully joined ===========


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

Running and Testing the Smart Contract

Vilnius University

Package and Install the Smart Contract (1)

- We need to **package a smart contract** before we can install it to the channel.
- Navigate to the **manufacturer/contract** folder directory and run the **npm install** command:
  - cd pharma-ledger-network/organizations/manufacturer/contract
  - npm install
- This will install the ***pharmaledgercontract*** node dependency under **node_modules/** folder

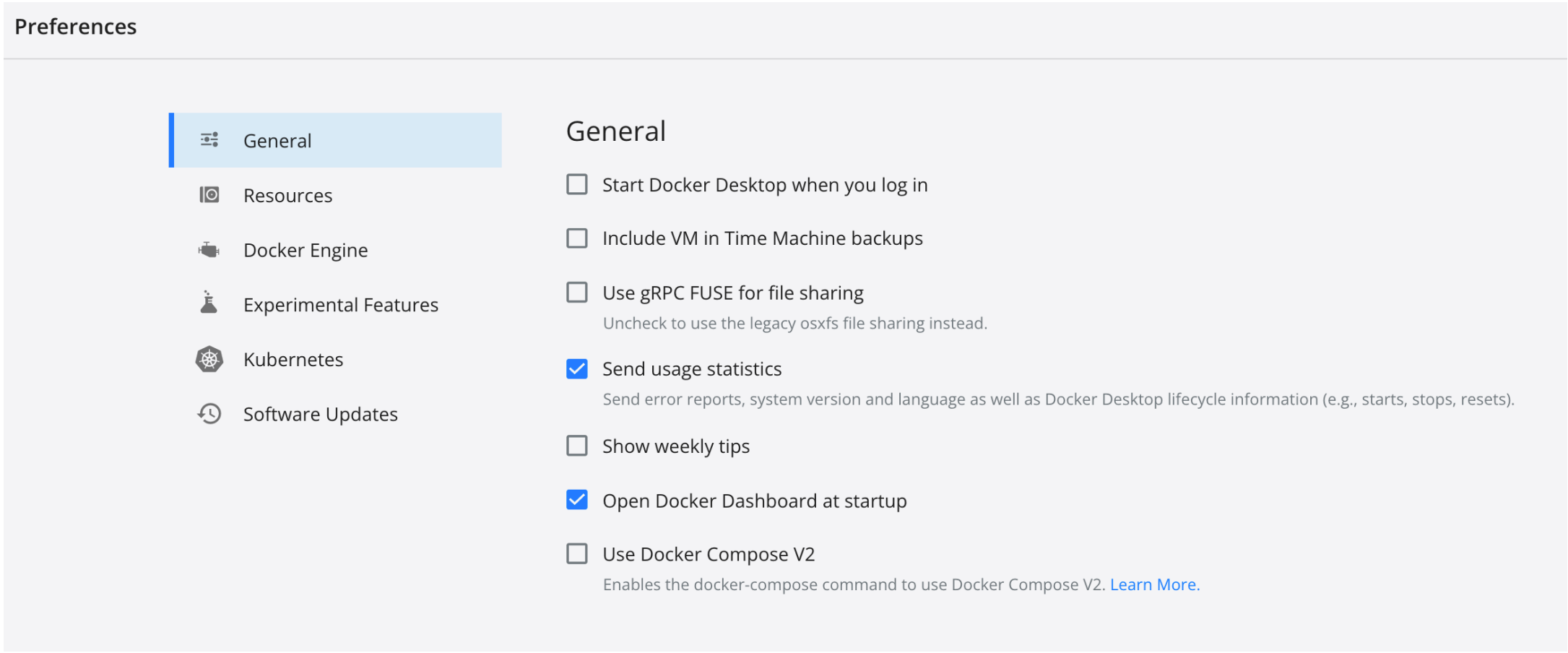
```
~/HyperFabric/pln-supply-chain/pharma-ledger-network/organizations/manufacturer/contract 15:26:28
$ ls
index.js          lib               node_modules      package-lock.json package.json
```

Package and Install the Smart Contract (2)

- Next, let's deploy/install the **pharmaledgercontract** chaincode in our PLN network by running the command: **./net-pln.sh deploySmartContract**
- If the command is successful, you should see the following in the last few lines

```
Using organization 3
***** [Step: 7]: start call queryCommitted org3 on peer: peer0.org3, channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 **
***
Attempting to Query committed status on peer0.org3, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID plnchannel --name pharmaLedgerContract
+ res=0
+ set +x

Committed chaincode definition for chaincode 'pharmaLedgerContract' on channel 'plnchannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true, Org3MSP: true]
***** completed call queryCommitted, Query committed on channel 'plnchannel' on channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, se
quence 1 *****

***** completed call queryAllCommitted, Chaincode installed on channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****

========= Pharma Ledger Network (PLN) contract successfully deployed on channel plnchannel  ===========


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

Troubleshooting Docker



Disabling "Use gRPC FUSE for file sharing" in the docker settings solved my issue.

Test the Smart Contract (1)

- We have created invokeContract.sh for this project.
- It defines an invocation method for **makeEquipment**, **wholesalerDistribute**, **pharmacyReceived**, and a **query** function.
- Now we can start testing our smart contract for these functions:
  1. **Call the makeEquipment chaincode method**: ./net-pln.sh invoke equipment GlobalEquipmentCorp 2000.001 e360Ventilator GlobalEquipmentCorp
     - You will see logs similar to the following:

```
+ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls true --cafile /Users/remigijus/HyperFabric/supply-chain/
pharma-ledger-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C plnchannel
 -n pharmaLedgerContract --peerAddresses localhost:7051 --tlsRootCertFiles /Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles /Users/remigijus/HyperFabric
/supply-chain/pharma-ledger-network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt --peerAddresses localhost:11
051 --tlsRootCertFiles /Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/peerOrganizations/org3.example.com/peers/peer0.org3.
example.com/tls/ca.crt -c '{"function":"makeEquipment","Args":["GlobalEquipmentCorp","2000.001", "e360Ventilator", "GlobalEquipmentCorp"]}'
+ res=0
+ set +x
2021-12-16 19:52:30.144 EET [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
***** completed call invokeMakeEquipment, Invoke transaction successful on channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, sequenc
e 1 *****
```

Test the Smart Contract (2)

2. After invoking **makeEquipment**, we can run a query function to verify the ledger result. The query function uses the **peer chaincode query command**: ./net-pln.sh invoke query 2000.001
   The query should return current equipment state data:

```
{"Key":"2000.001","Record":{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmentCorp","equipmentName":"e360Ventilator","ownerName":"GlobalEquipme
ntCorp","previousOwnerType":"MANUFACTURER","currentOwnerType":"MANUFACTURER","createDateTime":"Thu Dec 16 2021 17:52:30 GMT+0000 (Coordinated Universal T
ime)","lastUpdated":"Thu Dec 16 2021 17:52:30 GMT+0000 (Coordinated Universal Time)"}}
***** completed call chaincodeQuery, Query successful on channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
```

3. Continue to invoke the remaining equipment functions for the wholesaler and pharmacy:
   ./net-pln.sh invoke wholesaler 2000.001 GlobalWholesalerCorp
   ./net-pln.sh invoke pharmacy 2000.001 PharmacyCorp
4. Once equipment ownership is moved to the pharmacy, the supply chain reaches its final state. We can issue **queryHistoryByKey** from the peer chaincode query command. Let's check equipment historical data: ./net-pln.sh invoke queryHistory 2000.001

Test the Smart Contract (3)

- We can see the following output in the terminal:

```
***** start call chaincodeQueryHistory on peer: peer0.org1, channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
Attempting to Query peer0.org1, Retry after 3 seconds.
+ peer chaincode query -C plnchannel -n pharmaLedgerContract -c '{"function":"queryHistoryByKey","Args":["2000.001"]}'
+ res=0
+ set +x

[{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmentCorp","equipmentName":"e360Ventilator","ownerName":"PharmacyCorp","previousOwnerType":"WHOL
ESALER","currentOwnerType":"PHARMACY","createDateTime":"Thu Dec 16 2021 17:52:30 GMT+0000 (Coordinated Universal Time)","lastUpdated":"Thu Dec 16 2021 18
:09:36 GMT+0000 (Coordinated Universal Time)"},{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmentCorp","equipmentName":"e360Ventilator","owner
Name":"GlobalWholesalerCorp","previousOwnerType":"MANUFACTURER","currentOwnerType":"WHOLESALER","createDateTime":"Thu Dec 16 2021 17:52:30 GMT+0000 (Coor
dinated Universal Time)","lastUpdated":"Thu Dec 16 2021 18:09:12 GMT+0000 (Coordinated Universal Time)"},{"equipmentNumber":"2000.001","manufacturer":"Gl
obalEquipmentCorp","equipmentName":"e360Ventilator","ownerName":"GlobalEquipmentCorp","previousOwnerType":"MANUFACTURER","currentOwnerType":"MANUFACTURER
","createDateTime":"Thu Dec 16 2021 17:52:30 GMT+0000 (Coordinated Universal Time)","lastUpdated":"Thu Dec 16 2021 17:52:30 GMT+0000 (Coordinated Univers
al Time)"}]
***** completed call chaincodeQuery, Query History successful on channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
```

- All of the transaction history records are displayed as output.
- We have tested our smart contract, and it works as expected.

Developing an Application with Hyperledger Fabric Through the SDK

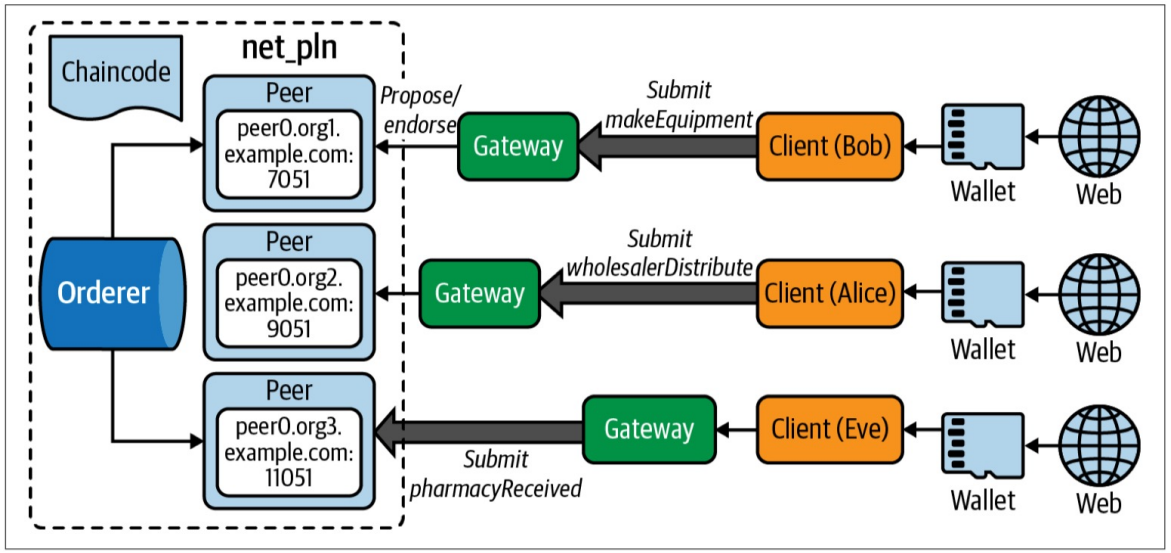
Vilnius University

Developing an Application with Hyperledger Fabric Through the SDK

- We just deployed our Pharma Ledger Network in the Fabric network.
- The next step is to build a **Pharma Ledger client application** to interact with the smart contract function in the network.
- Let's take a moment to examine the application architecture.
- At the beginning of our PLN network section, we generated a Common Connection Profile (CCP) for Org1, Org2, and Org3.
- We will use these connection files to connect to our PLN network for each peer org.
- When the manufacturer application's user Bob submits a makeEquipment transaction to the ledger, the pharma-ledger process flow starts.

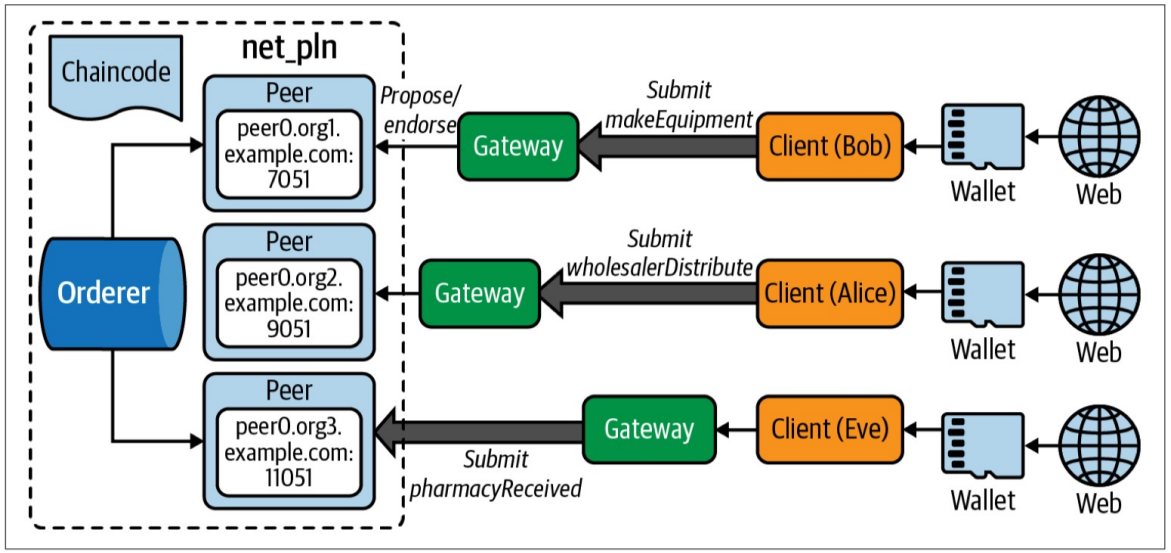
How the PLN application works (1)

- The **manufacturer** web user Bob **connects** to the Fabric network **through a wallet**.
- A **wallet provides users an authorized identity** that will be verified by the blockchain network to ensure access security.
- The Fabric SDK then submits a makeEquipment transaction proposal to peer0.org1.example.com.
- **Endorsing peers verify the signature, simulate the proposal, and invoke** the makeEquipment chaincode function with required arguments.



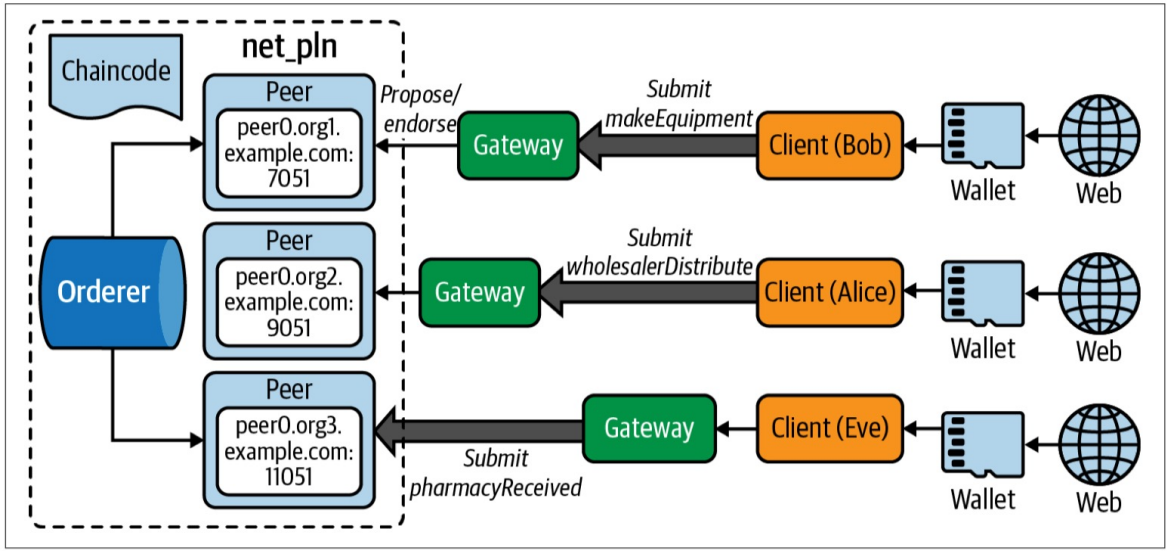
How the PLN application works (2)

- The **transaction is initiated** after the proposal response is sent back to the SDK.
- The **application collects and verifies the endorsements until the endorsement policy of the chaincode is satisfied with producing the same result**.
- The **client then broadcasts the transaction proposal** and **proposal response to the ordering service**.



How the PLN application works (3)

- The **ordering service orders them chronologically by channel**, **creates blocks, and delivers the blocks of transactions to all peers on the channel**.
- The **peers validate transactions to ensure that the endorsement policy is satisfied** and to ensure that no changes have occurred to the ledger state since the proposal response was generated by the transaction execution.
- After successful validation, the **block is committed to the ledger, and world states are updated for each valid transaction**.



Pharma-ledger client application (1)

- You now understand the transaction end-to-end workflow.
- It is time to start building our **pharma-ledger client application**.
- Figure shows the application client project structure.
- The same folder structure is available for the **wholesaler** and **pharmacy**.

```
$ tree -L 3
.
├── app.js
├── package.json
├── public
│   ├── css
│   │   └── style.css
│   ├── images
│   │   └── wait.gif
│   └── js
│       └── plnClient.js
├── services
│   ├── equipmentService.js
│   ├── queryHistoryService.js
│   ├── queryService.js
│   └── walletsService.js
└── views
    └── index.ejs
```

The application client project structure

Pharma-ledger client application (2)

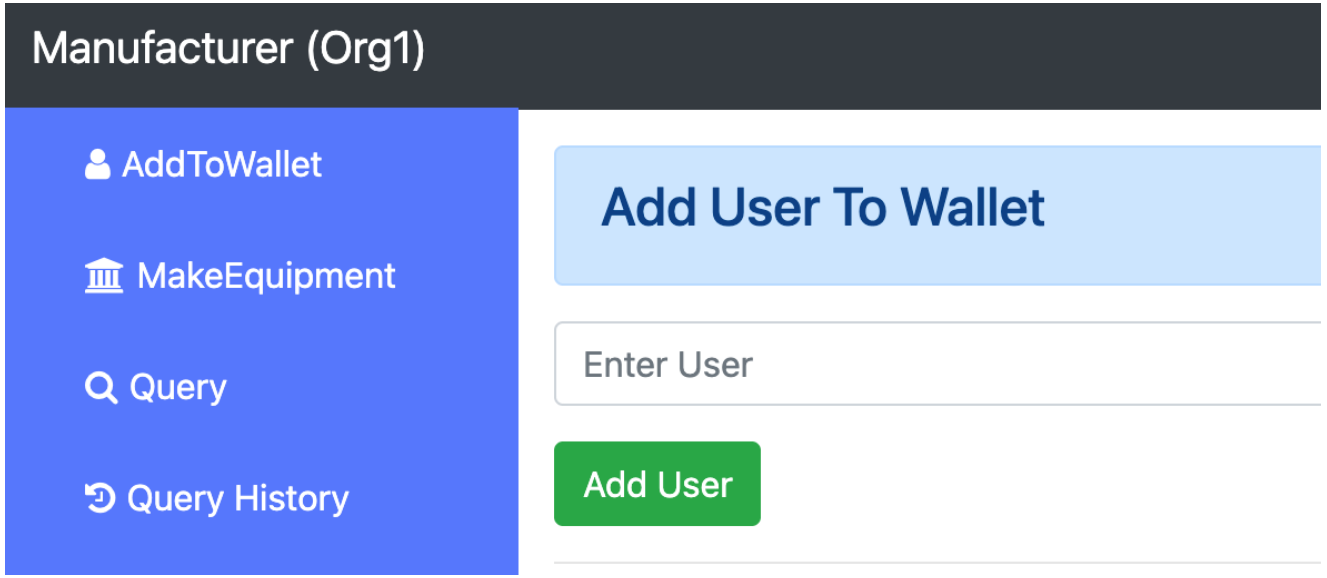
- Let's bring up a manufacturer, create the user Bob, and then submit a transaction to our PLN blockchain.
- Navigate to the **pharma-ledger-network/organizations/manufacturer/application** folder. Then:
  1. **Update the client IP address in plnClient.js file under public/js/**

```
9       var urlBase = "http://localhost:30000";
10      //var urlBase = "http://your-ip:30000";
```

  2. **Run: npm install**
  3. **Run: node app.js**
  4. **Open a browser and enter: http://localhost:30000/**

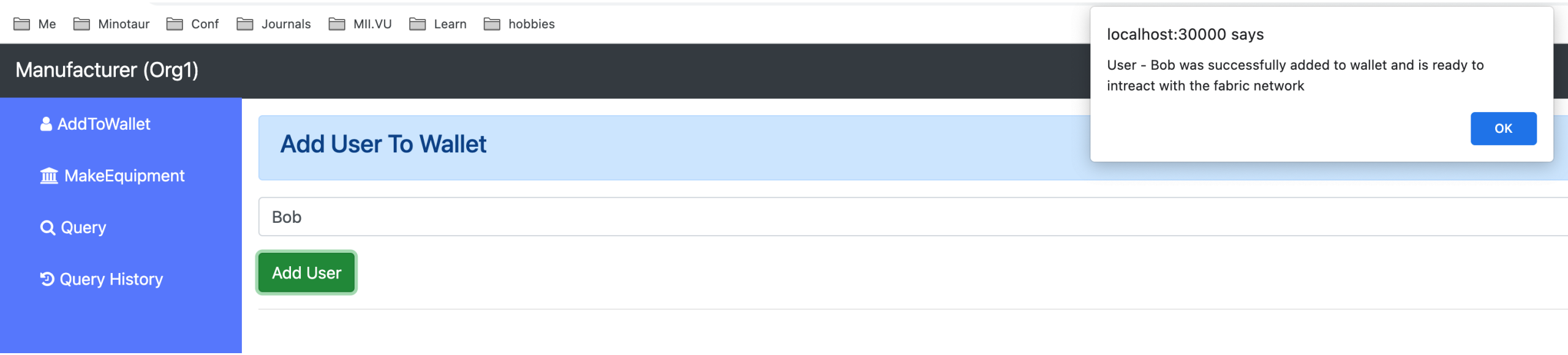
Pharma-ledger client application (3)

- We will see the screen shown in Figure
- The default page is **addToWallet**.
- Since we have not added any user to the wallet so far, you **can't submit makeEquipment** and **query history transactions** at this moment.
- You have to add a user to the wallet.



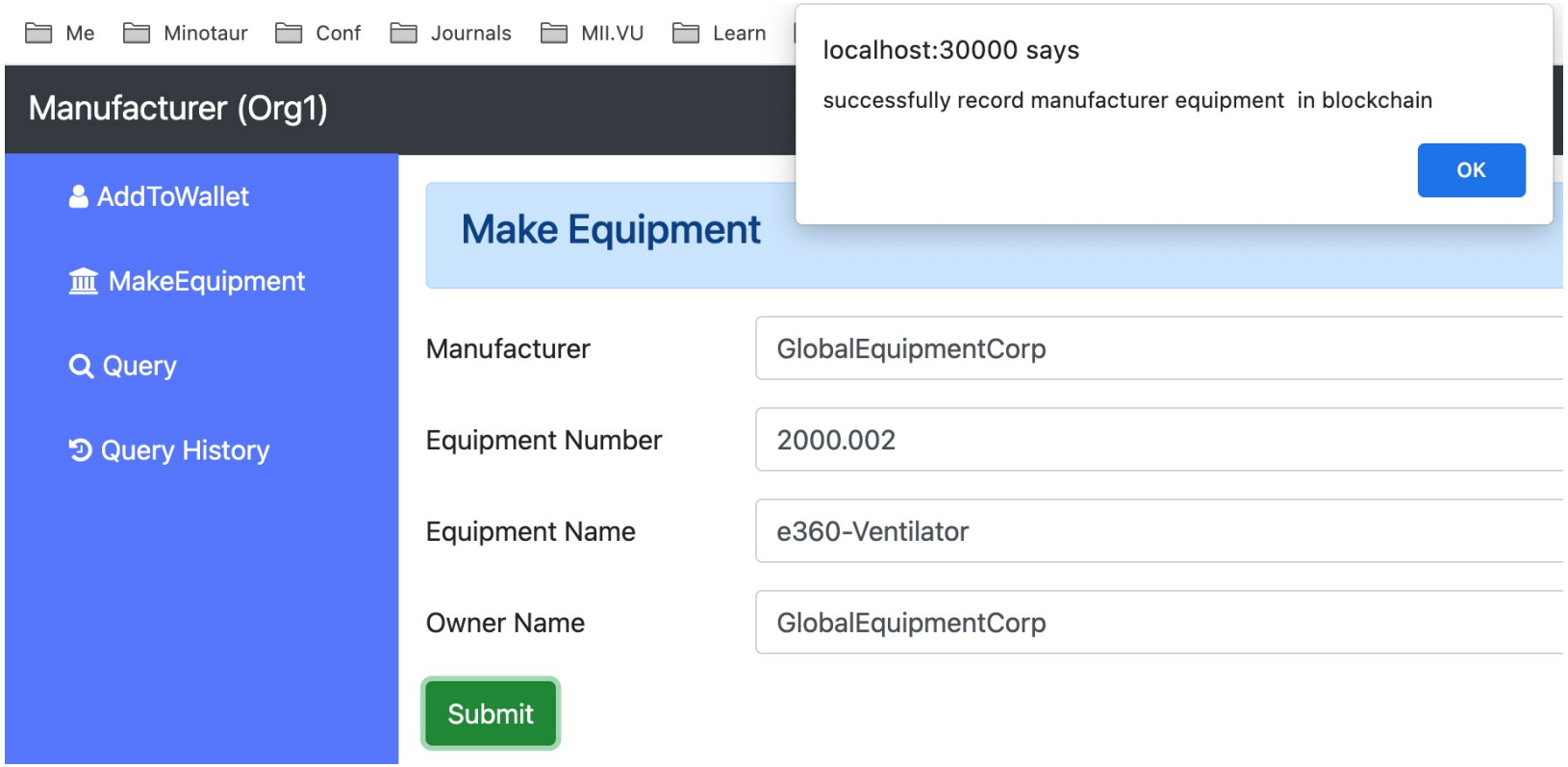
Pharma-ledger client application (4)

- Let’s add Bob as a manufacturer user, as shown in Figure
- With the user wallet set up, the application can now connect to our PLN and interact with the chaincode.



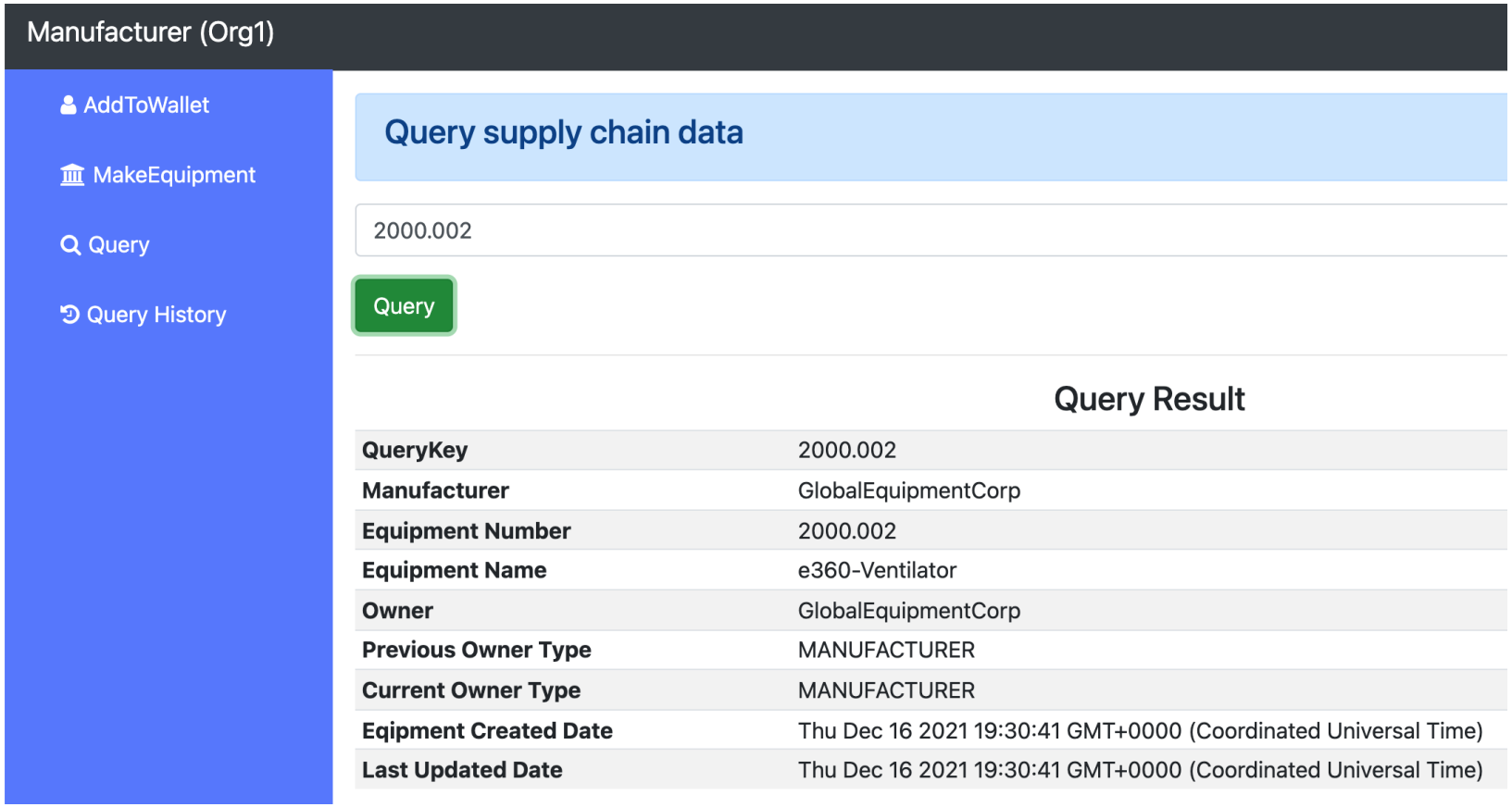
Pharma-ledger client application (5)

- Click **MakeEquipment** on the left menu, enter all required equipment information, and submit the request (see Figure).
- The success response will be returned from the blockchain.



Pharma-ledger client application (6)

- We can now **query** equipment data in the PLN network by equipment number.
- Figure shows the result



Pharma-ledger client application (7)

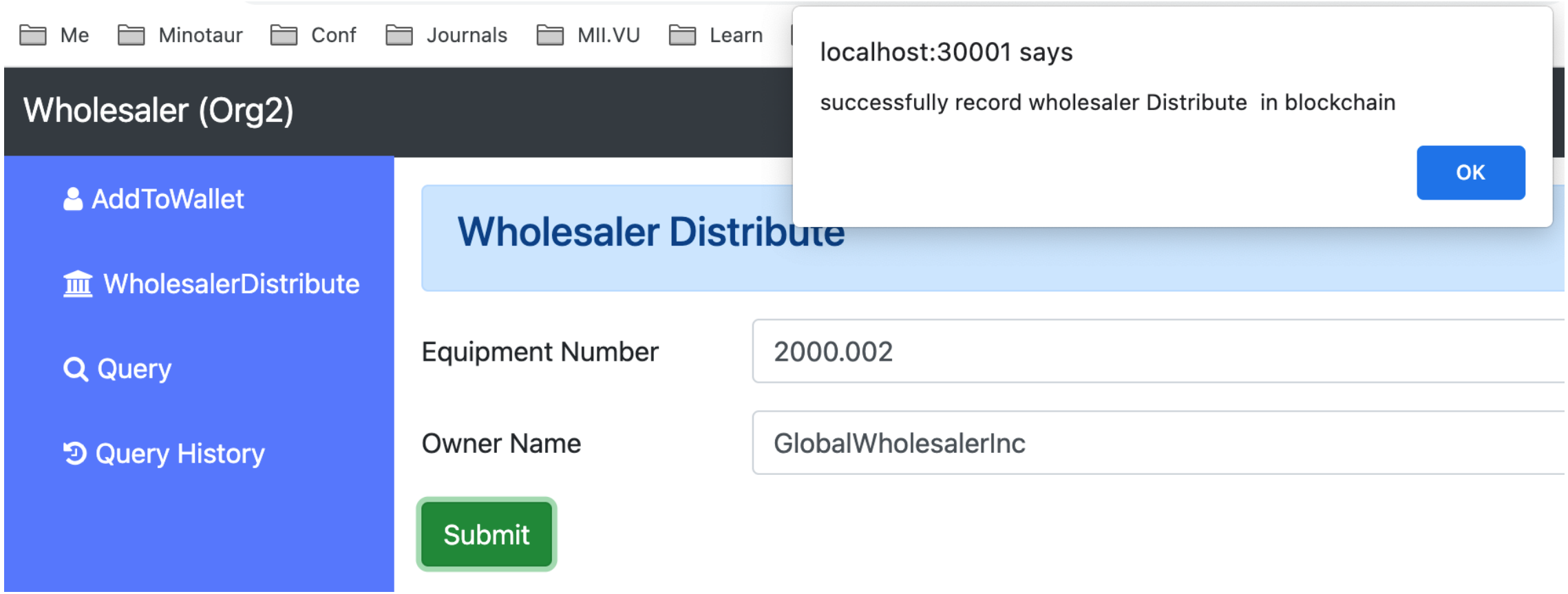
- Now **open two other terminal windows**, which will bring up node servers for the wholesaler and pharmacy, respectively.
- Navigate to **/pharma-ledger-network/organizations/wholesaler/application** folder. Then:
  1. **Update the client IP address in plnClient.js file under public/js/**

```
9       var urlBase = "http://localhost:30001";
10      //var urlBase = "http://54.161.38.96:30001";
```

  2. **Run: npm install**
  3. **Run: node app.js**
  4. Open a browser and enter: **http://localhost:30001/**

Pharma-ledger client application (8)

- Add Alice as a **wholesaler** user (Figure) and submit a **wholesalerDistribute** request



Pharma-ledger client application (9)

- Follow the same steps by bringing up the pharmacy node server
- Navigate to **/pharma-ledger-network/organizations/pharmacy/application** folder. Then:
  1. **Update the client IP address in plnClient.js file under public/js/**

```
9     var urlBase = "http://localhost:30002";
10    //var urlBase = "http://54.161.38.96:30002";
```

  2. **Run: npm install**
  3. **Run: node app.js**
  4. Open a browser and enter: **http://localhost:30002/**

```
~/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/application 21:56:42
$ node app.js                                                    ‹system›
node:internal/modules/cjs/loader:936
  throw err;
  ^

Error: Cannot find module 'fabric-contract-api'
Require stack:
- /Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/contract/lib/pharmaledgercontract.js
- /Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/application/services/pharmacyService.js
- /Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/application/app.js
    at Function.Module._resolveFilename (node:internal/modules/cjs/loader:933:15)
    at Function.Module._load (node:internal/modules/cjs/loader:778:27)
    at Module.require (node:internal/modules/cjs/loader:999:19)
    at require (node:internal/modules/cjs/helpers:102:18)
    at Object.<anonymous> (/Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/contract/lib/pharmaledgercontract.js:21:31)
    at Module._compile (node:internal/modules/cjs/loader:1095:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1147:10)
    at Module.load (node:internal/modules/cjs/loader:975:32)
    at Function.Module._load (node:internal/modules/cjs/loader:822:12)
    at Module.require (node:internal/modules/cjs/loader:999:19) {
  code: 'MODULE_NOT_FOUND',
  requireStack: [
    '/Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/contract/lib/pharmaledgercontract.js',
    '/Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/application/services/pharmacyService.js',
    '/Users/remigijus/HyperFabric/supply-chain/pharma-ledger-network/organizations/pharmacy/application/app.js'
  ]
}
```

Vilnius University

AČIŪ UŽ DĖMESĮ!

KONTAKTAI

Prof. dr. Remigijus Paulavičius,
remigijus.paulavicius@mif.vu.lt

Dr. Ernestas Filatovas:
ernestas.filatovas@mif.vu.lt