Blockchain Development
Week: 7
Title: Node.js and TX: Lists

Dr Ian Mitchell

Middlesex University,
Dept. of Computer Science,
London

September 26, 2019

Lecture Objectives

Knowledge

- Search
- Lists, Arrays
- UpdateAll
- Advanced JS - more promises
- Pizza Delivery
- Events
- Emit

Disclaimer

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

Approach

Mistakes

We can learn a lot from bad design. Sometimes it is necessary to make mistakes in order to learn. Here we look at implementation of Arrays of items in registries, however, care is to be taken in and warning signs should be given and the reduction in data redundancy is a good thing. Styles of programming will also be looked at, many coders avoid the use of promises and we will look at this approach.

Bad Examples

- Trader example
- keep tabs on trader commodities
- restricted view
- add trader
- remove trader

Scenario

- Each trader has a list of commodities they currently own
- For academic purposes
- **Consequences from last week**
- removing a member of staff was difficult. Why?

Scenario

- Each trader has a list of commodities they currently own
- For academic purposes
- **Consequences from last week**
- removing a member of staff was difficult. Why?
- Only the owner can sell assets
- if the member of staff removed had assets

Scenario

- Each trader has a list of commodities they currently own
- For academic purposes
- **Consequences from last week**
- removing a member of staff was difficult. Why?

- Only the owner can sell assets
- if the member of staff removed had assets
- these assets remain locked in, no one can sell them

Scenario

- Each trader has a list of commodities they currently own
- For academic purposes
- **Consequences from last week**
- removing a member of staff was difficult. Why?

- Only the owner can sell assets
- if the member of staff removed had assets
- these assets remain locked in, no one can sell them
- each trader keeps a lists of the assets they own
- requires updating each time a commodity changes ownership, for the seller and the buyer.

Scenario

- Each trader has a list of commodities they currently own
- For academic purposes
- **Consequences from last week**
- removing a member of staff was difficult. Why?

- Only the owner can sell assets
- if the member of staff removed had assets
- these assets remain locked in, no one can sell them
- each trader keeps a lists of the assets they own
- requires updating each time a commodity changes ownership, for the seller and the buyer.
- when a member of staff leaves a nominated member of staff is given all the assets, and then the member of staff is deleted.

Trader

CTO

```
/**
 * Sample business network definition.
 */
namespace org.t4.net

enum Grade {
  o manager
  o consultant
  o intern
  o clerk
}

asset Commodity identified by tradingSymbol {
  o String tradingSymbol
  o String description
  o Double quantity
  --> Trader owner
}

participant Trader identified by tradeId {
  o String tradeId
  o String firstName
  o String lastName
  o Grade Status
  o String[] commoditiesOwned
}

transaction Trade {
  --> Commodity commodity
  --> Trader newOwner
```

Difference from last week?

Trader

CTO

```
 1 /**
 2  * Sample business network definition.
 3  */
 4 namespace org.t4.net
 5 
 6 enum Grade {
 7   o manager
 8   o consultant
 9   o intern
10   o clerk
11 }
12 
13 asset Commodity identified by tradingSymbol {
14   o String tradingSymbol
15   o String description
16   o Double quantity
17   --> Trader owner
18 }
19 
20 participant Trader identified by tradeId {
21   o String tradeId
22   o String firstName
23   o String lastName
24   o Grade Status
25   o String[] commoditiesOwned
26 }
27 
28 transaction Trade {
29   --> Commodity commodity
30   --> Trader newOwner
```

Difference from last week?

- line 25 - Array
- Array is to represent all the commodities owned

Traders

JSON

```
{
  "$class": "org.t4.net.Trader",
  "tradeId": "0227",
  "firstName": "",
  "lastName": "",
  "Status": "manager",
  "commoditiesOwned": []
}
```

```
{
  "$class": "org.t4.net.Trader",
  "tradeId": "1711",
  "firstName": "",
  "lastName": "",
  "Status": "manager",
  "commoditiesOwned": [
    "8084",
    "7856",
    "8941",
    "2139",
    "2336"
  ]
}
```

Commodities

JSON

```
{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "2139",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "2336",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "7856",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "8084",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "8941",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}
```

Trade Transaction

- Check?

Trade Transaction

- Check?
- Buyer exists?
- Commodity exists?
- Updates?

Trade Transaction

- Check?
- Buyer exists?
- Commodity exists?
- Updates?
- Commodity ownership
- Trader: commoditiesOwned array

Trade Transaction

- Check?
- Buyer exists?
- Commodity exists?
- Updates?
- Commodity ownership
- Trader: commoditiesOwned array
- Buyer: Adding to array
- Seller: Removing from array

Trader Transaction

JS - Does Buyer Exist?

```
/**
 * transaction of a commodity from one trader to another
 * @param {org.t4.net.Trade} trade - the trade to be processed
 * @transaction
 */
async function tradeCommodity(tx) {
  var ns="org.t4.net";
    var me=getCurrentParticipant();
    var updateArray = new Array();

    return getParticipantRegistry(ns+".Trader")
      .then(function (traderRegistry){
      return traderRegistry.exists(tx.newOwner.getIdentifier())
      .then(function(exists){
```

Trader Transaction

JS - Add Commodity to Buyer Array

```
        if (exists){
            return traderRegistry.get(tx.newOwner.getIdentifier())
            .then(function(singleTrader){
// add the commodity id from the tx to the new owner in singleTrader
                singleTrader.commoditiesOwned.push(tx.commodity.getIdentifier().toString());
                console.log('Update newOwner after sale');
                updateArray.push(singleTrader);
```

Trader Transaction

JS - Remove Commodity to Seller Array

```
  // remove the commodity id from the tx from the old owner in traderRegistry
                  let needle=tx.commodity.tradingSymbol.toString();
                  let haystack=me.commoditiesOwned;
                  let filteredHaystack = haystack.filter((item)=>item!==needle);
                  me.commoditiesOwned = filteredHaystack;
                  updateArray.push(me);
   // update the trader registry using the updated Array of traders
                  traderRegistry.updateAll(updateArray);
```

Trader Transaction

JS - Commodity to Ownership updated

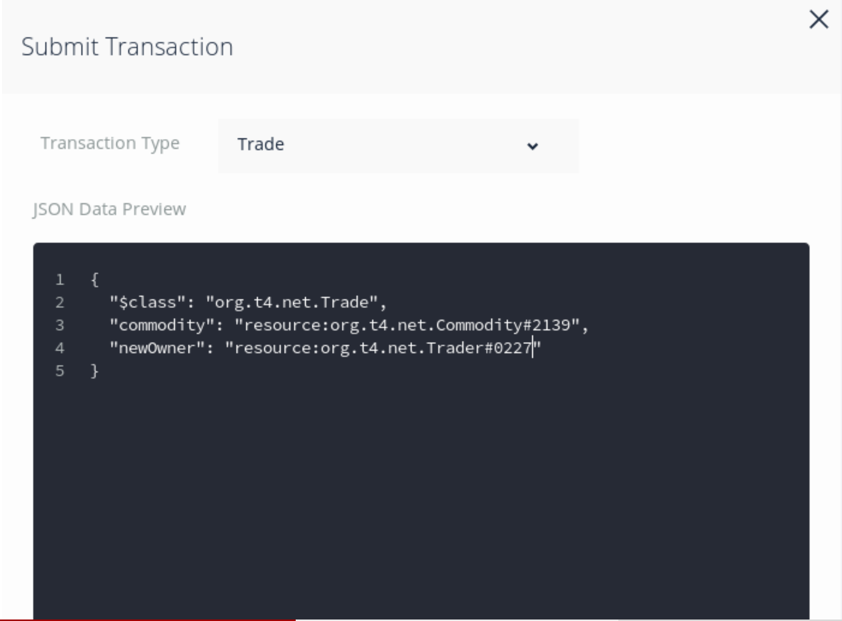
```
                  return getAssetRegistry(ns+'.Commodity')
              .then( function(commodityReg){
   // update the owner in the commodity
                        tx.commodity.owner=tx.newOwner;
                return commodityReg.update(tx.commodity);
                })
              })
          }
```

Trader Transaction

JS - Buyer does not exist

```
          else
          {
            throw new Error('New owner does not exist');
          }
      })
    })
}
```

Completing the Transaction



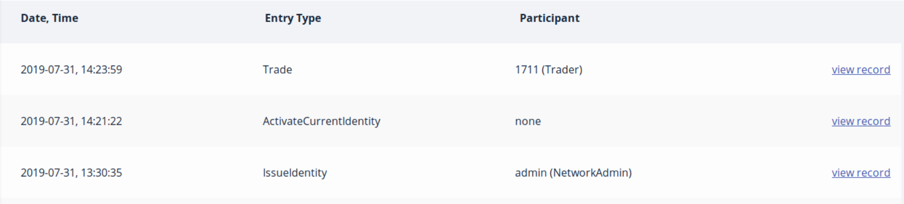
Submit Transaction

Transaction Type: Trade

JSON Data Preview

```
{
  "$class": "org.t4.net.Trade",
  "commodity": "resource:org.t4.net.Commodity#2139",
  "newOwner": "resource:org.t4.net.Trader#0227"
}
```

Transaction History



Date, Time	Entry Type	Participant	
2019-07-31, 14:23:59	Trade	1711 (Trader)	view record
2019-07-31, 14:21:22	ActivateCurrentIdentity	none	view record
2019-07-31, 13:30:35	IssueIdentity	admin (NetworkAdmin)	view record

Transaction History block

```
{
 "$class": "org.t4.net.Trade",
 "commodity": "resource:org.t4.net.Commodity#2139",
 "newOwner": "resource:org.t4.net.Trader#0227",
 "transactionId": "474a3060-b46e-469f-a1cb-51838fe6c0bf",
 "timestamp": "2019-07-31T13:23:59.548Z"
}
```

Traders

JSON

```
{
  "$class": "org.t4.net.Trader",
  "tradeId": "0227",
  "firstName": "",
  "lastName": "",
  "Status": "manager",
  "commoditiesOwned": [
    "2139"
  ]
}
```

```
{
  "$class": "org.t4.net.Trader",
  "tradeId": "1711",
  "firstName": "",
  "lastName": "",
  "Status": "manager",
  "commoditiesOwned": [
    "8084",
    "7856",
    "8941",
    "2336"
  ]
}
```

Commodities

JSON

```
{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "2139",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#0227"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "2336",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "7856",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "8084",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}

{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "8941",
  "description": "",
  "quantity": 0,
  "owner": "resource:org.t4.net.Trader#1711"
}
```

Alternative

- Pizza Delivery
- with a promise chain
- using await command
- The burden of access is shifted. Where?

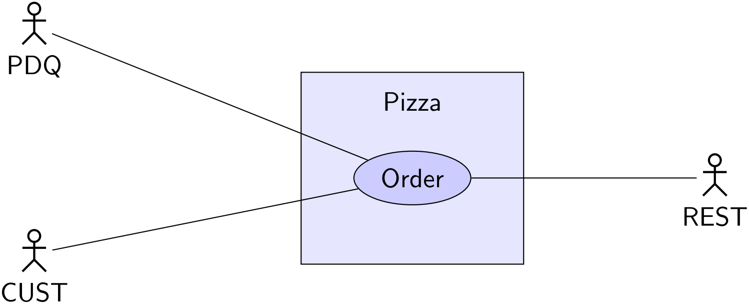
- Look at examples on github `https://github.com/hyperledger/composer-sample-networks`
- These cannot be used for the coursework.
- Pizza

Alternative

- Pizza Delivery
- with a promise chain
- using await command
- The burden of access is shifted. Where?
- The burden is shifted from JS to ACL
- Look at examples on github `https://github.com/hyperledger/composer-sample-networks`
- These cannot be used for the coursework.
- Pizza

Pizza

Simplified Use Case



Pizza

CTO - Status

```

/* ENUMERATOR */
enum STATUS {
  o PLACED
  o PREPARED
  o DISPATCHED
  o DELIVERED
```

Pizza

CTO - Status

```

/* ENUMERATOR */
enum STATUS {
  o PLACED
  o PREPARED
  o DISPATCHED
  o DELIVERED
```

- lifecycle of order
- PLACED - create order by customer
- PREPARED - update by pizzOutlet
- DISPATCHED - update by pizzaOutlet
- DELIVERED - update by pizzaOutlet

Pizza

CTO - Size & PizzaType

```
}
enum SIZE {
  o small
  o medium
  o large
}
enum PIZZATYPE{
  o americana
  o carbonara
  o margherita
  o marinara
  o napoli
  o quattro
  o romana
```

- Toppings
- Size
- Pizza Type
- Enumerator Types

Pizza

CTO - Address - Customer - Restaurant

```
}
/* CONCEPT */
concept ADDRESS{
  o String Name optional
  o String NameNumber default="1"
  o String Street default="High St"
  o String PostCode default="NW44BT"
}

/* PARTITICPANT */
participant customer identified by customerID{
  o String customerID
  o ADDRESS deliveryAddress
}

participant pizzaOutlet identified by poID{
  o String poID
  o ADDRESS poAddress
}

participant pqc identified by pqcID{
  o String pqcID
```

Pizza

CTO - Order

```
}
/* current version only allows 1 pizza per order
 * simply rectified by adding array
 * --> pizzaDetail[] pizzas
*/
asset order identified by orderID{
  o String orderID
  --> pizzaDetail pizza
  --> pizzaOutlet restaurant
  --> customer consumer
  o STATUS status
```

- Where does ID come from?

Pizza

CTO - Order

```
}
/* current version only allows 1 pizza per order
 * simply rectified by adding array
 * --> pizzaDetail[] pizzas
*/
asset order identified by orderID{
  o String orderID
  --> pizzaDetail pizza
  --> pizzaOutlet restaurant
  --> customer consumer
  o STATUS status
```

- Where does ID come from?
- User generated, can be pseudo-random
- Comment on mulitple orders
- array of pizzaDetails
- TOPPING is inaccessible
- Usually an order has 3 things:

Pizza

CTO - Order

```
}
/* current version only allows 1 pizza per order
 * simply rectified by adding array
 * --> pizzaDetail[] pizzas
*/
asset order identified by orderID{
  o String orderID
  --> pizzaDetail pizza
  --> pizzaOutlet restaurant
  --> customer consumer
  o STATUS status
```

- Where does ID come from?
- User generated, can be pseudo-random
- Comment on mulitple orders
- array of pizzaDetails
- TOPPING is inaccessible
- Usually an order has 3 things:
  1. Product: Pizza, sometimes the quantity
  2. Seller: Restaurant
  3. Buyer: Customer
- STATUS: track progress

Pizza

CTO - Transactions & Events

```
transaction prepareOrder{
  --> order pizzaPrepared
}

event prepareOrderEvent{
  --> order pizzaPrepared
}

transaction dispatchOrder{
  --> order pizzaDispatched
}

event dispatchOrderEvent{
  --> order pizzaDispatched
}

transaction deliverOrder{
  --> order pizzaDelivered
}

event deliverOrderEvent{
  --> order pizzaDelivered
}
```

CustomerSeeSelf: Customers can only see themselves

Rules

ACL - Customer

```
rule customerSeeSelf{
  description: "customer see themselves"
  participant(p): "org.pqc.uk.customer"
  operation: ALL
  resource(r): "org.pqc.uk.customer"
  condition: (p.getIdentifier()==r.
      getIdentifier())
  action: ALLOW
}
rule customerSeePizza{
  description: "customer see pizza"
  participant: "org.pqc.uk.customer"
  operation:READ
  resource: "org.pqc.uk.pizzaDetail"
  action: ALLOW
}
rule customerSeeOrder{
  description: "customer see pizza"
  participant(p): "org.pqc.uk.customer"
  operation:ALL
  resource(r): "org.pqc.uk.order"
  //transaction(t): "org.pqc.uk.placeOrder"
  condition: (p.getIdentifier()==r.consumer.
      getIdentifier())
  action: ALLOW
}
```

CustomerSeeSelf: Customers can only see themselves. Condition that ensures the consumer in the order is equal to the customer.

CustomerSeePizza: Customers can see the pizzas available

Rules

ACL - Customer

```
rule customerPlaceOrder{
  description: "customer places order"
  participant: "org.pqc.uk.customer"
  operation: ALL
  resource: "org.pqc.uk.placeOrder"
  action: ALLOW
}
rule customerReadRestaurant{
  description: "customer has read access to restaurants"
  participant: "org.pqc.uk.customer"
  operation: READ
  resource: "org.pqc.uk.pizzaOutlet"
  action: ALLOW
}
```

customerPlaceOrder:

Only a customer can place an order and access transaction `placeOrder`

customerReadRestaurant:

Customers are permitted to read pizzaOutlet details

Rules

ACL - Restaurant

```
rule restaurantSeeSelf{
  description: "restaurants can only view their own details"
  participant(p): "org.pqc.uk.pizzaOutlet"
  operation: ALL
  resource(r): "org.pqc.uk.pizzaOutlet"
  condition: (p.getIdentifier()==r.getIdentifier())
  action: ALLOW
}
rule restaurantSeeOrders{
  description: "restaurant can only see their own orders"
  participant(p): "org.pqc.uk.pizzaOutlet"
  operation: ALL
  resource(r): "org.pqc.uk.order"
  condition: (p.getIdentifier()==r.restaurant.getIdentifier())
  action:ALLOW
}
```

restaurantSeeSelf:

Restaurant can only see themselves

restaurantSeeOrders:

Restaurant can only see orders placed at their pizzaOutlet

Rules

ACL - Restaurant

```
rule restaurantReadsCustomer{
  description: "restaurant reads customer"
  participant: "org.pqc.uk.pizzaOutlet"
  operation: READ
  resource: "org.pqc.uk.customer"
  action:ALLOW
}
rule restaurantPlaceOrder{
  description: "restaurant reads order"
  participant: "org.pqc.uk.pizzaOutlet"
  operation: READ, UPDATE//CANNOT CREATE
  resource: "org.pqc.uk.order"
  transaction: "org.pqc.uk.prepareOrder"
  action: ALLOW
}
rule restaurantProcessOrder{
  description: "restaurant process order"
  participant: "org.pqc.uk.pizzaOutlet"
  operation: ALL
  resource: "org.pqc.uk.prepareOrder"
  action: ALLOW
```

restaurantReadsCustomer:
restaurant can read customer details

restaurantPlaceOrder:
Restaurants cannot place orders, merely read and update the status of them

restaurantProcessOrder:
Restaurants can process orders from status `PLACED` to `PREPARED` using transaction `prepareOrder`

Rules

ACL - Restaurant

```
rule restaurantDispatchOrder{
  description: "restaurant dispatch order access"
  participant: "org.pqc.uk.pizzaOutlet"
  operation:ALL
  resource:"org.pqc.uk.dispatchOrder"
  action:ALLOW
}
rule restuarantDeliverOrder{
  description: "restaurant deliver order access"
  participant: "org.pqc.uk.pizzaOutlet"
  operation:ALL
  resource: "org.pqc.uk.deliverOrder"
  action:ALLOW
}
```

restaurantDispatchOrder:
: Restaurant can process orders from status `PREPARED` to `DISPATCHED` using transaction `restaurantDispatchOrder`

restaurantDeliverOrder:
: Restaurant can process orders from status `DISPATCHED` to `DELIVERED` using the transaction `restaurantDeliverOrder`

Transactions

JS - Place Order

```
/*
 * User submits order to restaurant
 * @param {org.pqc.uk.placeOrder} placeOrder - pizza order
 * @transaction
 */
async function placeOrder(tx){
    const ns='org.pqc.uk';
//create new order
    var factory = getFactory();
    var newOrder=factory.newResource(ns,'order',tx.orderID);
    newOrder.pizza = tx.pizza;
    newOrder.restaurant = tx.restaurant;
    newOrder.consumer = tx.Customer;
    newOrder.status = 'PLACED';
// add new order to the order registry
    const orderReg = await getAssetRegistry(ns+'.order');
    await orderReg.add(newOrder);
}
```

Transactions

JS - Prepare Order

```
/*
 * restaurant prepares order
 * @param {org.pqc.uk.prepareOrder} prepareOrder - pizza order
 * @transaction
 */
async function prepareOrder(tx){
    const ns=’org.pqc.uk’;
    currentOrder = tx.pizzaPrepared;
    if( currentOrder.status !== ’PLACED’)
    {
        throw new Error(’Current order’+currentOrder.orderID+’ is in wrong status to be prepared’);
    }
    else
    {
        currentOrder.status = ’PREPARED’;
    }
// update order with currentOrder
    const orderReg = await getAssetRegistry(ns+’.order’);
    await orderReg.update(currentOrder);
// emit the event
    const factory=getFactory();
    const prepareOrderEvent=factory.newEvent(ns,’prepareOrderEvent’);
    prepareOrderEvent.pizzaPrepared=currentOrder;
    emit(prepareOrderEvent);
}
```

Transactions

JS - Dispatch Order

```
/*
 * restaurant dispatches order
 * @param{org.pqc.uk.dispatchOrder} dispatchOrder - pizza dispatched
 * @transaction
 */
async function dispatchOrder(tx){
    const ns='org.pqc.uk';
    Prepare currentOrder=tx.pizzaDispatched;
    if( currentOrder.status !== 'PREPARED')
    {
        throw new Error('Current order has not been prepared');
    }
    else
    {
        currentOrder.status = 'DISPATCHED';
    }
// update order with currentOrder
    const orderReg = await getAssetRegistry(ns+'.order');
    await orderReg.update(currentOrder);
// emit the event
    const factory=getFactory();
    const dispatchOrderEvent=factory.newEvent(ns,'dispatchOrderEvent');
    dispatchOrderEvent.pizzaDispatched = currentOrder;
    emit(dispatchOrderEvent);
}
```

Transactions

JS - Deliver Order

```
/*
 * customer receives order
 * @param{org.pqc.uk.deliverOrder} deliverOrder - pizza delivered
 * @transaction
 */
async function deliverOrder(tx){
    const ns='org.pqc.uk';
    currentOrder=tx.pizzaDelivered;
    if( currentOrder.status !== 'DISPATCHED')
    {
        throw new Error('Current order has not been dispatched');
    }
    else
    {
        currentOrder.status = 'DELIVERED';
    }
// update order with currentOrder
    const orderReg = await getAssetRegistry(ns+'.order');
    await orderReg.update(currentOrder);
// emit the event
    const factory=getFactory();
    const deliverOrderEvent=factory.newEvent(ns,'deliverOrderEvent');
    deliverOrderEvent.pizzaDelivered=currentOrder;
    emit(deliverOrderEvent);
}
```

References I

[1] Nitin Gaur et al. *Hands-on Blockchain with Hyperledger: Building Decentralised Applications with Hyperledger Fabric and Composer*. Packt, 2018. ISBN: 9781788994521.

[2] *Hyperledger Architecture, Volume 1*. 2017.

[3] *Hyperledger Architecture, Volume 2*. 2018.

Web Resources

- http://hyperledger.org
- https://nodejs.org
- https://hyperledger.github.io/composer/latest/api/runtime-factory
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
- https://github.com/hyperledger/composer-sample-networks
- https://hyperledger.github.io/composer/latest/business-network/bnd-create