Blockchain Development
Week: 6
Title: Node.js and TX

Dr Ian Mitchell

Middlesex University,
Dept. of Computer Science,
London

September 26, 2019

Lecture Aims

Aims

There are four components to hyperledger's composer playground:

1. Data
2. Access
3. Logic
4. Query

After last week's introduction to Node.js, this week we will investigate how node.js can be applied to fit the logic of a blockchain application, and essentially work towards transaction completion.

Lecture Objectives

Knowledge

- Retrieving registers
- Updating registers
- Transaction Completion
- Composer API

Hyperledger
Composer

- ACL
- CTO
- Javascript ES6
  - ECMAScript 2015
  - ECMA is a an organisation for the standardisation of information and communication systems
  - ECMA was founded in 1961
  - `www.ecma-international.org`
- Netscape submitted Javascript to ECMA for standardisation
- Result ECMAScript
- ES1 - 1997
- ES6 - 2015
- ES8 - 2017

Hyperledger Composer API
Registry Classes

- AssetRegistry
- ParticipantRegistry
- IdentityRegistry
- TransactionRegistry
- Historian

Registry functions

- Add one or more items
- Update one or more items
- Remove one or all items
- Get one or all items
- Check if an item exists
- Resolve one or all items

Transactions

lib/logic.js

header

```
/**
* Create a transaction
* @param {namespace.TransactionName} tx - further comment
* @transaction
*/
```

Trader [1] I

CTO

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

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
}

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

[1]adapted from hyperledger.org tutorials

Trader [2] I

ACL

```
/**
 * Sample access control list.
 */

rule SystemACL {
    description: "System ACL to permit all access"
    participant: "org.hyperledger.composer.system.Participant"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

[2]adapted from hyperledger.org tutorials

Trader [3] I
JS

```
/**
 * transaction of a commodity from one trader to another
 * This check could be completed with ACL and is an exercise
 * @param {org.t4.net.Trade} trade - the trade to be processed
 * @transaction
 */
async function tradeCommodity(tx) {
  var ns="org.t4.net.";
    tx.commodity.owner=tx.newOwner;
    const commodityRegister = await getAssetRegistry(ns+"Commodity");
    await commodityRegister.update(tx.commodity);
}
```

[3] adapted from `hyperledger.org` tutorials

Compare CTO and JS

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

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
}

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

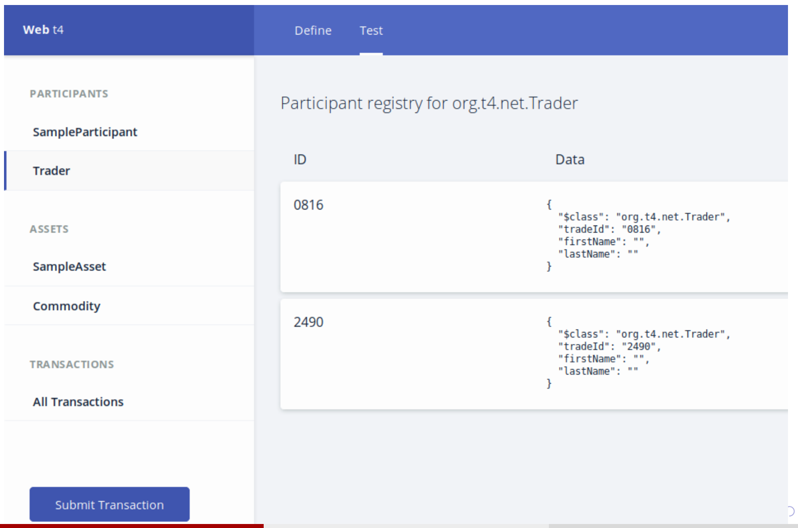
```
/**
 * transaction of a commodity from one
     trader to another
 * This check could be completed with ACL
     and is an exercise
 * @param {org.t4.net.Trade} trade - the
     trade to be processed
 * @transaction
 */
async function tradeCommodity(tx) {
  var ns="org.t4.net.";
    tx.commodity.owner=tx.newOwner;
    const commodityRegister = await
     getAssetRegistry(ns+"Commodity");
    await commodityRegister.update(tx.
     commodity);
}
```

Traders Example

```
1  /**
2   * transaction of a commodity from one
        trader to another
3   * This check could be completed with ACL
        and is an exercise
4   * @param {org.t4.net.Trade} trade - the
        trade to be processed
5   * @transaction
6   */
7  async function tradeCommodity(tx) {
8    var ns="org.t4.net.";
9      tx.commodity.owner=tx.newOwner;
10     const commodityRegister = await
        getAssetRegistry(ns+"Commodity");
11     await commodityRegister.update(tx.
        commodity);
12 }
```

- 1-5 comments
- @param has namespace, followed by transaction name
- @transaction identifies the following function as a TX
- 6 function name is unique and does not match transaction name. It does take transaction as a parameter.
- 7 changes ownership, owner replaced by newOwner
- 8 get Commodity registry
- 9 update Commodity registry

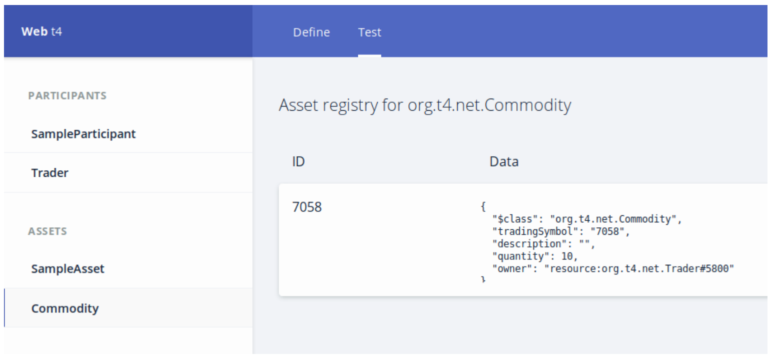
Traders Example
Output Trader



Participant registry for org.t4.net.Trader

ID	Data
0816	{ "$class": "org.t4.net.Trader", "tradeId": "0816", "firstName": "", "lastName": "" }
2490	{ "$class": "org.t4.net.Trader", "tradeId": "2490", "firstName": "", "lastName": "" }

Traders Example
Output Commodity



Asset registry for org.t4.net.Commodity

ID	Data
7058	{ "$class": "org.t4.net.Commodity", "tradingSymbol": "7058", "description": "", "quantity": 10, "owner": "resource:org.t4.net.Trader#5800" }

Traders Comparison?

Traders

```
{
  "$class": "org.t4.net.Trader",
  "tradeId": "0816",
  "firstName": "",
  "lastName": ""
}
{
  "$class": "org.t4.net.Trader",
  "tradeId": "2490",
  "firstName": "",
  "lastName": ""
}
```

Commodity

```
{
  "$class": "org.t4.net.Commodity",
  "tradingSymbol": "7058",
  "description": "",
  "quantity": 10,
  "owner": "resource:org.t4.net.Trader#5800"
}
```

Exists

```
/**
 * 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 newOwner = tx.newOwner;
    return getParticipantRegistry(ns+"Trader")
      .then(function (traderRegistry){
          return traderRegistry.exists(newOwner.getIdentifier());
      })
      .then(function (exists){
          if (exists){
              tx.commodity.owner=tx.newOwner;
                return getAssetRegistry(ns+"Commodity")
                  .then( function(commodityRegistry){
                      return commodityRegistry.update(tx.commodity)
                })
            }
          else
            {
              throw new Error("New Owner, "+newOwner.getIdentifier()+", does not exist as
      a Trader. Enter an existing new Owner");
            }
    })
}
```

Exists

- use of promise chains
- first get all traders
- then call method exists
- if exists evaluates to true, the trader exists
- the get asset registry
- and update
- else throw an error

Exist

- inherited from Registry
- Determines whether a specific resource exists
- Returns - a promise that will be resolved with true or false depending on whether the resource exists

Particpant Registry Methods

SuperType	Name	Return	Description
Registry	add	Promise	Adds a new resource to the registry
Registry	addAll	Promise	Adds a list of new resources to the reg istry
Registry	exists	Promise	Determines whether a specific resource exists in the registry
Registry	get	Promise	Get a specific resource in the registry
Registry	getAll	Promise	Get all the resources in the registry
Registry	remove	Promise	Remove a resource with a given id fron the registry
Registry	removeAll	Promise	Remove a list of resources from the reg istry

Particpant Registry Methods

SuperType	Name	Return	Description
Registry	resolve	Promise	Get a specific resource in the registry, and resolve all of the relationships to othe assets, participants and transactions
Registry	resolveAll	Promise	Get all the resources in the registry and resolve all their relationships to other as- sets, participants and transactions
Registry	update	Promise	Updates a resource in the registry
Registry	updateAll	Promise	Updates a list of resources in the registry

Add

Scenario

- Create a transaction that allows managers to add and remove staff
- Check the status of the current participant
- Create a new participant
- Then allow them to add a participant to the registry

Requirements

- status for participant
- need factory to create a new resource
- use of add method

Factory Methods

Name	Return	Description
newConcept	Concept	Creates a new concept with a given namespace, type and identifier
newEvent	Resource	Create a new type with a given names- pace and type
newRelationship	Relationship	Create a new relationship with a given namespace, type and identifier
newResource	Resource	Create a new resource (an instance of an asset, participant or transaction)

Add
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 }
```

⋮

```
32 transaction AddStaff{
33   o Trader newTrader
34 }
```

Add
JS

```
27 /*
28  * transaction to add new member of staff
29  * @param {org.t4.net.AddStaff} AddStaff - add new staff
30  * @transaction
31  */
32 async function addNewStaff(tx){
33   var ns1='org.t4.net';
34   var me=getCurrentParticipant();
35   if (me.Status === 'manager') {
36     return getParticipantRegistry(ns1+'.Trader')
37       .then( function(traderRegistry){
38             //console.log('me.status value: '+me.Status);
39             var factory = getFactory();
40             var newStaff = factory.newResource(ns1, 'Trader', tx.newTrader.tradeId);
41             console.log('new factory: complete');
42             newStaff=tx.newTrader;
43           console.log('new trader:'+newStaff.tradeId);
44             return traderRegistry.add(newStaff);
45     })
46     }
47     else
48     {
49           throw new Error("You have insufficient privileges to add a member of staff");
50     }
51 }
```

Add Method reviewed

- Notice namespace variable declared differently
- See enumerator type in CTO
- Note transaction AddStaff in CTO
- Mainly to do with using both Factory and Registry methods
- Ideally, have a global variable for namespace
- triple equals sign
- console.log
- allows for debugging
- different browsers have different methods for display
- Currently, Firefox uses CTRL+I
- newStaff is created on lines 39-40
- newStaff is populated on line 42
- Finally, added to the trade registry on line 44

Remove
CTO

⋮

```
transaction RemoveStaff{
  o Trader removedStaff
}
```

Remove
JS

⋮

```
/*
 * transaction to remove staff from registry
 * @param {org.t4.net.RemoveStaff} RemoveStaff - remove staff member
 * @transaction
 */
async function removeStaff(tx){
  var ns='org.t4.net';
  var me=getCurrentParticipant();
  if (me.Status==='manager'){
    return getParticipantRegistry(ns+'.Trader')
    .then( function(traderRegistry){
      var factory=getFactory();
      var leavingStaff=factory.newResource(ns,'Trader',tx.removedStaff.tradeId);
      leavingStaff=tx.removedStaff;
      return traderRegistry.remove(leavingStaff);
    })
  }
  else
  {
    throw new Error('Insufficient Privileges: Managers only to remove staff');
  }
}
```

Review Method reviewed

- Exactly same as Add
- namespace
- get Current participant
- test for status
- if match to manager then remove
- else display error message
- get the appropriate registry
- create factory
- pass the values from the transaction variable to the factory
- remove the staff entry from the registry
  - All so easy?

Review Method reviewed

- Exactly same as Add
- namespace
- get Current participant
- test for status
- if match to manager then remove
- else display error message
- get the appropriate registry
- create factory
- pass the values from the transaction variable to the factory
- remove the staff entry from the registry
  - All so easy?
  - Any commodities the removed Staff member is in ownership?
  - How does this impact the trading scenario
  - Is there a solution?

Summary

- Promises
- Transactions
- Get Registry
- Update Registry
- Add Registry
- Exist Registry
- Get Current Participant
- Console Log
- Errors

References I

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

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

Web Resources

- `http://hyperledger.org`
- `https://nodejs.org`
- `https://hyperledger.github.io/composer/latest/api/runtime-factory`