CST4125: Blockchain Development
Week: 6
Title: Node.js and TX

Dr Ian Mitchell

Bedfordshire,
UK

2023

Lecture Aims

Aims

There are four components to hyperledger's composer playground:

1. Data
2. Access
3. Logic

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

Contact and Office Hours

Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: `smerf.net`

Contact and Office Hours

Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: `smerf.net`

Office Hours

- During term time only
- When: Autumn Term: Mondays 1100-1300hrs
- Please read notifications or emails
- There are occassions that these could be arranged online, e.g., due to industrial action or inclement weather

Deadlines

				Feedback	
Description	Submission	Weight	Deadline	Formative	Summative
1. Hyperledger	MyLearning	50%	18th December 2022	LW11-12	12/01/2023
2. Ethereum	MyLearning	50%	2nd April 2023	LW23-24	24/04/2023
Resits	MyLearning	50-100%	1st July 2023	None	None
Deferals	MyLearning	50-100%	1st July 2023	None	None

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);
}
```

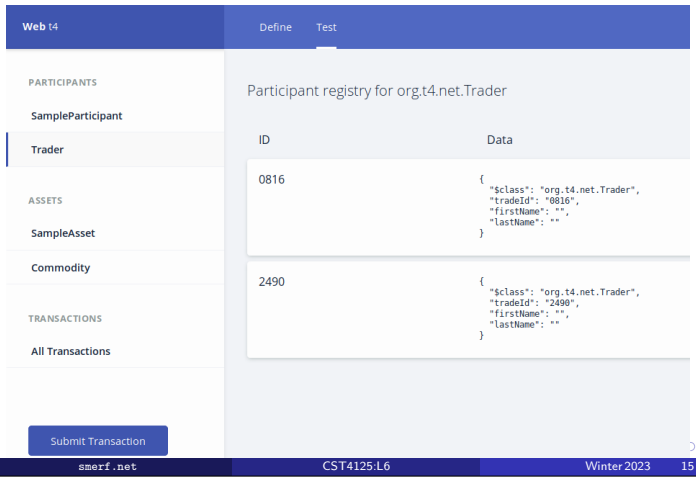

Traders Example

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

```
/**
 * 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);
}
```

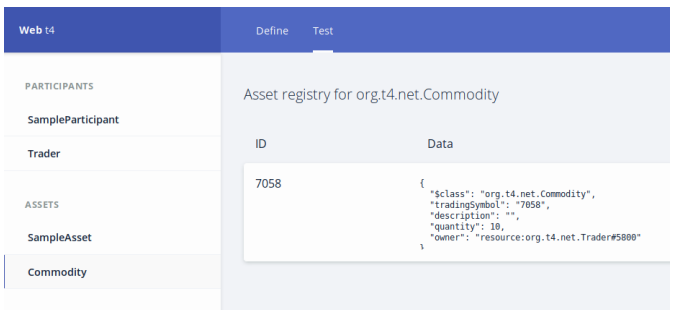

Traders Example

Output Trader



Traders Example

Output Commodity



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
- then 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 registry
Registry	exists	Promise	Determines whether a specific resourceexists 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 from the registry
Registry	removeAll	Promise	Remove a list of resources from the registry

Particpant Registry Methods

SuperType	Name	Return	Description
Registry	resolve	Promise	Get a specific resource in the registry, and resolve all of the relationships to other assets, participants and transactions
Registry	resolveAll	Promise	Get all the resources in the registry and resolve all their relationships to other assets, 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 namespace 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

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

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

...

transaction AddStaff{
  o Trader newTrader
}
```


Add
JS

```
/*
 * transaction to add new member of staff
 * @param {org.t4.net.AddStaff} AddStaff - add new staff
 * @transaction
 */
async function addNewStaff(tx){
  var ns1='org.t4.net';
  var me=getCurrentParticipant();
  if (me.Status === 'manager') {
    return getParticipantRegistry(ns1+'.Trader')
      .then( function(traderRegistry){
            //console.log('me.status value: '+me.Status);
            var factory = getFactory();
            var newStaff = factory.newResource(ns1, 'Trader', tx.newTrader.tradeId);
            console.log('new factory: complete');
            newStaff=tx.newTrader;
          console.log('new trader:'+newStaff.tradeId);
            return traderRegistry.add(newStaff);
    })
    }
    else
    {
          throw new Error("You have insufficient privileges to add a member of staff");
    }
}
```


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
- newStaff is populated
- Finally, added to the trade registry

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?

Change Status

- Change status of assets during transactions
- Various checks regarding state
- If true then update state
- Effects of changing state, identify any dependencies

Reflection

New TX

1. ACL
2. Add TX to DM
3. Add wallets
4. Write Code
5. Test

Access Rules

- Access Rules embedded in TX code
- Access Rules in ACL
- Always in ACL
- Maintenance
- Separation of concerns

Practice v. Theory

No Promise Chain

```
const ns='org.t2.net';
/**
 * transaction of a commodity from one trader to another
 * @param {org.t2.net.Trade} trade - the trade to be processed
 * @transaction
 */
async function tradeCommodity(tx) {
  let traderRegistry = await getParticipantRegistry(ns+'.Trader');
  let traderExists = await traderRegistry.exists(tx.newOwner.getIdentifier());
  if(traderExists){
      tx.commodity.owner = tx.newOwner;
    let assetRegistry = await getAssetRegistry(ns+'.Commodity');
      await assetRegistry.update(tx.commodity);
  } else {
    throw new Error(" New Owner " + tx.newOwner.getIdentifier() + " does not exist");
  }
}
```


Practice v. Theory

Integrated rules

```
/**
* transaction to add a new member of staff - manager access only
* @param {org.t2.net.AddStaff} tx - one param
* @transaction
*/
async function addNewStaff(tx) {
  let me=getCurrentParticipant();
  if (me.role==='Manager')
  {
    var traderReg= await getParticipantRegistry(ns+'.Trader');
    await traderReg.add(tx.newStaff);
  }
  else
  {
    throw new Error("You have insufficient privileges");
  }
}
```


Practice v. Theory

Integrated rules

```

rule addNewStaff{
  description: "Manager adds new staff"
  participant: "org.t2.net.Trader"
  operation: ALL
  resource: "org.t2.net.Trader"
  action:ALLOW
}

rule addNewStafftx{
  description: "Manager to access tx"
  participant: "org.t2.net.Trader"
  operation: ALL
  resource: "org.t2.net.AddStaff"
  action:ALLOW
}
```

Practice v. Theory

Separate rules

```
/**
* transaction to remove a new member of staff - manager access only
* @param {org.t2.net.removeStaff} tx - one param
* @transaction
*/
async function removeStaff(tx) {
    var traderReg= await getParticipantRegistry(ns+'.Trader');
    await traderReg.remove(tx.removedStaff);
}
```


Practice v. Theory

Separate rules

```
rule removeStaff{
  description: "Manager remove staff"
  participant(p): "org.t2.net.Trader"
  operation: ALL
  resource(r): "org.t2.net.Trader"
  condition: (p.role==="Manager")
  action: ALLOW
}
rule removeStafftx{
  description: "Manager remove staff tx"
  participant(p): "org.t2.net.Trader"
  operation: ALL
  resource(r): "org.t2.net.removeStaff"
  condition: (p.role==="Manager")
  action: ALLOW
}
```


Summary

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

References I

[1] The Linux Foundation. *Hyperledger Architecture, Volume 1*. hyperlink. [Accessed: Jan 2021]. 2017.

[2] The Linux Foundation. *Hyperledger Architecture, Volume 2*. hyperlink. [Accessed: Jan 2021]. 2018.

Web Resources

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