BLOCKCHAINS IN BUSINESS NETWORKS

A PROJECT REPORT

submitted by

YADUGIRI SAIKUMAR
EE14B067

in partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
in
ELECTRICAL ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
MAY 2018

REPORT CERTIFICATE

This is to certify that the report titled **Blockchains in Business Networks**, submitted by **Yadugiri Saikumar**, to the Indian Institute of Technology Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr Shweta Agrawal
Project Guide,
Assistant Professor,
Department of Computer Science and Engineering,
IIT Madras, 600 036.

Dr Krishna Jagannathan
Project Co-Guide,
Assistant Professor,
Department of Electrical Engineering,
IIT Madras, 600 036.

Place: Chennai,
Date: 5th May, 2018.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my guide Dr Shweta Agrawal for giving me an opportunity to work under her. Also I would like to thank you for constantly guiding me thoughtfully and efficiently throughout this project, giving me an opportunity to work at my own pace along my own lines, while providing me with very useful directions and insights whenever necessary. I would like to thank my Co-Guide for providing me essential support with my project.

I would also take this opportunity to thank all my friends who have been a great source of motivation and encouragement.

Finally I would also like to thank all of them who have helped me complete my project successfully.

Yadugiri Saikumar
EE14B067,
Undergraduate Student,
Department of Electrical Engineering,
IIT Madras, 600 036.

Place: Chennai,
Date: 5th May, 2018.

TABLE OF CONTENTS

Contents

Blockchains in Business Networks

Yadugiri Saikumar

3rd May, 2018, Indian Institute of Technology Madras

Abstract

Blockchains are distributed ledger services invented Satoshi Nakamoto for his infamous cryptocurrency Bitcoin. But the use cases of blockchains are enormous and can be used as a ledger system for almost all kinds of purposes. One of such fields is in business networks. In any business network may it be intra-organisation or inter-organisation, it needs faster computations and never lose the trust in the ledger system. One of the cryptographic ways of building trust is explored in the working of blockchains. In my B.Tech project I explore this idea and create two business network demos. One network contains of details of undergraduate students pursuing Electrical Engineering in IIT Madras and other is a land record data which needs constant updates using virtual data I generated.

Keywords: Blockchains, Hyperledger Composer, Hyperledger Fabric, AngularJS, Javascript

1. Introduction

A business network can be described as a group of organisations working together to achieve a common goal. If the network is intra-organisation, it can be thought of as a few departments working together. But some of the major similarities of any business network is the presence of participants who can be customers or suppliers or partners and some kind of wealth. The wealth can be monetary or non-monetary. Then there would be a market where transactions for the wealth take place. These markets can be private or public.

But in a digital world like ours, these business networks have taken their place in the Internet. One of the main advantages of such a transition is the speed provided by the computational resources. But speed does not necessarily mean rigidness and resiliency. So, there needs to be an entity which takes care of documenting all the changes to any entity in the network. Previously, it was done using a physical ledger of all the entities. This is a great idea to audit all the changes and declare the current values. But a business network need not only contain transactions. It can also contain contracts which in some cases are valid up to years. In such a scenario, having multiple ledgers can result in mistakes and can be time consuming to maintain. In addition if there a multiple participants in a network, all of them need not trust each other and maintain their own separate ledgers which is a waste of many resources. The key to solving these problems lie in the heart of blockchains.

The idea of having a shared, replicated and permissioned ledger can be intimidating to use at the first hear but the working and the underlying cryptographic concepts make sure that these are safe and easy to use and understand. These are called blockchains. A blockchain as the name suggests, is a growing chain of blocks which contains details required to store in the ledger. These are secured and linked through cryptographic hash functions. The very first block contains “null” values and is called as the genesis block of the blockchain. Blockchains are secure by design and provide distributed computational resources using Byzantine fault tolerance. In other words, there is no need for a third party to make sure that the information in the blocks is genuine. Though the idea of decentralised consensus is envisaged as early as 1994, it was put into practice for the cryptocurrency Bitcoin invented by an anonymous inventor under the name Satoshi Nakamoto. He discussed in his paper, *Bitcoin: A peer-to-peer Electronic Cash System*, the essentials of the operation of blockchains and their role in eliminating the double spending problem in bitcoins.

Though blockchains are conceived for bitcoins, their use in business networks need a separate definition. We have take into consideration that one of the main use of blockchains in bitcoin network, anonymity, should be given up for identification of the network participants. The consensus mechanism for bitcoin blockchains is "proof of work", in which the network tries to solve a hard problem through random oracle access to a cryptographic function. In a business network, we need selective endorsement as the consensus mechanism. Based on such different uses, we can say that blockchains underpin bitcoin.

2. Requirement for Blockchains in Business Networks

Four main advantages of using blockchains in business networks are -

1. They provide a shared ledger among the participants of the network. It is same ledger which records all the transactions across the business network. It is replicated and permissioned. So, the participant with limited permissions on the ledger can do only limited activities on the ledger.
2. They provide smart contract system. These are automated autonomous programs that reside on the blockchain and encapsulate business logic and code in order to execute a required function when certain conditions are met.
3. The ledger is a trusted source. The network decides who will endorse which transaction. These transactions cannot be edited or deleted or inserted. So, all the wealth will have a defined audit. Thus, they provide consensus, provenance and immutability.
4. The element of privacy. Transactions are needed to be authenticated but the person's identity need not be revealed when the authentication is done. Among a subset of participants, identity need not be linked in the transaction.

Other benefits include reduction of overheads such as space overhead and as the consensus mechanism is changed, transaction approval time can be mere seconds. As the transactions cannot be modified, there will be a clear record of all actions of participants. So, this increases trust and reduces risk.

2.1. Permissioned Blockchain Networks

One of the debatable things is the privacy. What are the aspects that need to be private and what should be disclosed? This is one of the reasons why we use permissioned setting in the network. In other words, we use a permissioned blockchain in place of a public blockchain like the bitcoin network. Permissioned or consortium or federated blockchains are the blockchains in which a certain group of people control the entry and exit into the group by a certificate in the genesis block of the chain. Easy way to understand the difference between the three types of blockchain networks is to understand the contrast among them. This is provided in the Table 2.1.

So, it is evident that the permissioned setting is the best way to implement a business network using blockchains. There are various open source projects available which explore such blockchain network settings. Some of the popular ones are Hyperledger, Multichain, Hydrachain etc. After experimenting with a few of them, I found out that Hyperledger suits all of our needs and it is constantly updated with new features in github. So, I developed 2 business networks with Hyperledger.

Public Blockchain Network	**Permissioned Blockchain Network**	**Private Blockchain Network**
Anyone can enter and exit the network freely.	Entrance and removal of a participant is decided by a group of existing members	Only a certain groups of people exist. Entry is restricted through invites or possession of the specified genesis block.
Every user contributes his computational power to every application in the network.	Certain group of people contribute for a required application dictated by the genesis block which in turn is decided by the group.	Only a certain group of people exist and only a set of applications are done.
It is highly computation exhaustive.	It is computationally resourceful.	It is computationally very fast and secure.
All the users have the same genesis block.	Genesis block has a different certificate for different group which can be modified.	
The identities of the users are private and all the transactions are public.	The identities of the users can be known through the certificates in genesis block and transactions can remain private.	All the identities are known and transactions are public.
Used in cryptocurrencies like Bitcoin etc.	Used in consortia.	Used in internal association transactions and ledgers.

Table 1: Differences between public, permissioned and private blockchain networks

3. Hyperledger

Hyperledger is an umbrella project for open-source blockchain development and other tools. It was started by The Linux Foundation Group in collaboration with IBM. It provides modular solutions for developing projects using blockchains. This is a token-less blockchain unlike Ethereum. Hyperledger itself is a suite of more than 6 different projects when it was launched in December 2016. Currently, it is still growing and including many more projects to serve different needs to various institutions such as the financial sector, retail, insurance and manufacturing. I have focused on two projects under Hyperledger which are Hyperledger Fabric and Hyperledger Composer. Hyperledger Fabric focuses on the architecture of the blockchain where as Hyperledger Composer is used to develop the details of the business network and deploy it using the Fabric's blockchain architecture.

3.1. Hyperledger Fabric

The main components of Hyperledger Fabric are -

- **Channel:** The simplest definition for a channel can be that it is private blockchain overlay. All the participants in a channel have a channel specific ledger. If there is any transaction, then it is sure that the peers endorsing it are the ones who have proper access to the channel.
- **Members:** These are entities which are participating in the network. These can be individuals or group of individuals or organisations.

- **Peers:** Peers are nodes which belong to a single organisation and which help in endorsing and committing transactions.
- **Orderer:** The orderer node collects the information about all the transactions. It then arranges them and verifies their validness.
- **Certificate Authority:** As permissioned setting is used in the network, it is the responsibility of this entity to distribute certificates which contains information about the role others are playing in the network.
- **Distributed Ledger:** The chain of blocks containing the transactions which is distributed and replicated among the peers of the network.
- **Chaincode:** This is the agent responsible to take care of the smart contract system. It contains information about all the assets and the information required to change the value of assets using transactions.
- **World State:** The state of the ledger which tells the current values of all the assets. It is useful for auditing the assets.
- **Endorsement Policy:** It is the policy which dictates which of the peer nodes should endorse the transaction as the "proof of work" is now changed to selective endorsement. The policy can be that a minimum number of peers must endorse or a specified group of peers must endorse etc.

For my project, I chose Hyperledger Fabric version 1. This is a stable release of the Hyperledger Fabric project. The consensus algorithm for this is selective endorsement. We use the endorsement policy to decide which peers vote. This method has a certain pros and cons of its own. The pros are that as this is a permissioned setting, it makes sense to use such a mechanism for consensus and thus reducing the time of transaction to a few seconds. It also provides crash fault tolerance. But this is not Byzantine fault tolerance. So, in the presence of any malicious nodes, the finality in the transaction may not be reached properly.

The transaction flow is simple. The chaincode with a certain endorsement policy is installed on all the peers in the network. When a request for a transaction is issued, based on the endorsement policy, certain peers are notified and they check the validity of the transaction. Furthermore, they check if the transaction has not already been proposed in the past to protect against the replay-attack and also make sure that the party which issued the transaction is properly authorised and his signature is valid, using MSP. When each and everyone of these are verified, the application broadcasts the transaction to the ordering service which orders the transaction and then updates the ledger which is immutable from that point.

3.2. Hyperledger Composer

Hyperledger Composer provides frameworks and toolkit to develop business applications based on blockchains. It is a suite of high-level application abstractions for business networks. It emphasises on business-centric vocabulary. It supports the existing Hyperledger Fabric and provides pluggable blockchain consensus protocol so that the end user has simple and controlled access points. It's end product provides a REST(Representational State Transfer) server API over which a foreign application can be developed such as an Angular application which I have developed. Using Hyperledger Composer saves time due to its various levels of abstractions. As the project is well tested and efficiently laid out, it is safe to use and it provides flexibility and it is easy to comprehend. Following the material in [3], I have developed two separate business networks. The first one using the students data from IITM Institute Web and Mobile Operations of which I was a part of. I then used a Java program(Students.java) to modify the data to fit my needs. I then generated a land data using Java program(Land.java) and all the related names and data are generated randomly from the Internet. I also generated the required transactions necessary for the land network using Java. I have enclosed these files in the report document. The following section contains details of the two networks I developed and the Angular websites for the easy use of those websites.

4. Network Development Details

The system configuration I used is as follows -

1. Operating System - Ubuntu 16.04 LTS 64-bit
2. curl version 7.47
3. Docker Engine version 18.03
4. Docker-Compose version 1.13
5. node version 8.10
6. npm version 5.7.1
7. Git version 2.17
8. Python version 2.7.12
9. Sublime 3.0 version 3143

These can also be installed using a Bash file(.sh file) from the official website, [3]. I have included the same along with the report. After installing all the necessary pre-requisites, to install Hyperledger Composer and Hyperledger Fabric, we can follow the official installation process for development environment in [3]. As Hyperledger Composer provides local and cloud service, it is better to try it in cloud first using [3] and its `Try it Online` option. But after we have installed Hyperledger Compose and Fabric, we can start the online bluemix project locally using the command

```
 $ composer-playground
```

Open `http://localhost:8080` in a browser and we can find same bluemix project found in the `Try it Online` option but it is running locally.

To begin with the development locally, we need to start Hyperledger Fabric first. Change the working directory to the folder where Hyperledger Fabric's `fabric-tools/` was installed. There we can see a bunch of Bash files. Execute startFabric.sh and createPeerAdminCard.sh using the commands

```
$ ./startFabric.sh
$ ./createPeerAdminCard.sh
```

Here in the console, we can see that we are using the Hyperledger Fabric version 1, `hlfv1`, stable version for development.

```
$./startFabric.sh
Development only script for Hyperledger Fabric control
Running 'startFabric.sh'
FABRIC_VERSION is unset, assuming hlfv1
...
```

To start a new business network, use the command

```
$ yo hyperledger-composer:businessnetwork
```

Then enter the details required such as name of the network, namespace and license. The license is used to generate a REST server API that is used in hosting the Angular website. As Apache provides an open source REST solution, we select the same. Namespace of the organisation is used to identify one in the network. It is necessary to be unique and also it is used to develop a website around it. So, it is recommended to show similarities in the namespace of the organisation and organisation's domain name. The names of the business networks I created are `students-network`, `land-record-network`. The main aspects that are needed to build the business network are

1. Model the Network
2. Transaction Logic for the Network
3. Creating Permissions for the Network

These are done in the following subsections.

4.1. Modelling the Network

After creation of the network, change the working directory to that network's folder. There we can see `models/` folder and inside it, we can see a `⟨organisation_namespace⟩.cto` file which is used to model our network. A Business Network contains of tangible or intangible assets, participants who own the assets or who are willing to participate in some transactions and the transactions too. These details are modelled in this file using an Object Oriented approach. The language in which they are written is the Hyperledger Composer Modelling Language. Full reference for this language can be found in [3] and in its `Install Hyperledger Composer` field. The assets and participants can be fully defined classes with their parameter in themselves. On the other hand, transactions are functions/methods and their parameters are specified in the model file. The following are the model files I used for the Students Network and the Land Record Network -

Listing 1: Model file for Students Network

```
namespace com.ac.iitm

asset Student identified by rollNo
{
  o String rollNo
  o String name
  o Integer roomNo
  o String hostel
  o String gender
}

participant Viewer identified by rollNo
{
  o String rollNo
  o String name
}
participant Staff identified by staffId
{
  o String staffId
  o String name
}

transaction ChangeName
{
  o String name
  o String relatedStudentRollNo
}
transaction ChangeHostel
{
  o String hostel
  o Integer roomNo
  o String relatedStudentRollNo
}
transaction ChangeGender
{
  o String gender
  o String relatedStudentRollNo
}
```

Listing 2: Model file for Land Record Network

```
namespace com.ac.iitm

participant LandOwner identified by ownerId
{
        o String ownerId
        o String name

}

asset Land identified by landId
{
        o String landId
        o String country
        o String ownerId
}

transaction LandTransaction
{
        o String idOfLand
        o String newOwnerId
}
```

The `identified by` field is basically the identifier of the certain asset or participant. It is compulsory to have the identifier as `String`. It makes sense to see that the identifier should be unique. But there is a bug in the project itself where we can change the identifier and after that, the corresponding asset will not be accessible by the old identifier or the new identifier.

4.2. Adding Transaction Logic to the Network

We have already defined the transaction requirements in the model file(.cto file). The actual logic of the transaction is defined using Javascript in the folder `lib/` and the file logic.js. The complete documentation required for transaction flow can be found in [4]. Here, we can also import certain rules using permissions.acl file which we can create separately. This is an Access Control File which is the core of the permissioned setting and allows access control to participants about the assets. We can also generate events which notify the participants about any assets they might be interested in or over which they have access over. There are three transactions defined which changes different fields of the asset. The logic for the transactions are as follows

Listing 3: Transaction Logic file for Land Record Network

```
'use strict';

/**
 * Transaction for changing name of student
 * @param {com.ac.iitm.ChangeName} changeName
 * @transaction
 */
function onChangeName(changeName)
{
    var assetRegistry;
    var id = changeName.relatedStudentRollNo;
    return getAssetRegistry('com.ac.iitm.Student').then(function(ar)
    {
        assetRegistry = ar;
        return assetRegistry.get(id);
    }).then(function(asset)
    {
        asset.name = changeName.name;
        return assetRegistry.update(asset);
    });
}

/**
 * Transaction for changing gender of student
 * @param {com.ac.iitm.ChangeGender} changeGender
 * @transaction
 */
function onChangeGender(changeGender)
{
    var assetRegistry;
    var id = changeGender.relatedStudentRollNo;
    return getAssetRegistry('com.ac.iitm.Student').then(function(ar)
    {
        assetRegistry = ar;
        return assetRegistry.get(id);
    }).then(function(asset)
    {
        asset.gender = changeGender.gender;
        return assetRegistry.update(asset);
    });
}

/**
 * Transaction for changing hostel of student
 * @param {com.ac.iitm.ChangeHostel} changeHostel
 * @transaction
 */
function onChangeHostel(changeHostel)
{
    var assetRegistry;
    var id = changeHostel.relatedStudentRollNo;
    return getAssetRegistry('com.ac.iitm.Student').then(function(ar)
    {
        assetRegistry = ar;
        return assetRegistry.get(id);
    }).then(function(asset)
    {
        asset.hostel = changeHostel.hostel;
        asset.roomNo = changeHostel.roomNo;
        return assetRegistry.update(asset);
    });
}
```

The basic structure to all the transactions is the same. All the parameters for the transaction are taken as a single dictionary given to the function. Then we are calling the registry of all the corresponding assets or participants using `getAssetRegistry()` or `getParticipantRegistry()` respectively. Then we are finding the concerned asset using its unique identifier, in the case of students, it is their roll number. Then we change the required field and update the asset. Then the function returns a `Promise` object which can be processed by the SDK to update the ledger through specified Hyperledger Fabric transaction flow. The following is the transaction logic for Land Record Network.

Listing 4: Transaction Logic file for Land Record Network

```
'use strict';

/**
 * Transaction for a land transaction involving a buyer and a seller.
 * @param {com.ac.iitm.LandTransaction} landTransaction
 * @transaction
 */
function onLandTransaction(landTransaction)
{
    var landRegistry;
    var id = landTransaction.idOfLand;
    return getAssetRegistry('com.ac.iitm.Land').then(function(ar)
    {
        landRegistry = ar;
        return landRegistry.get(id);
    }).then(function(land)
    {
        land.ownerId = landTransaction.newOwnerId;
        landRegistry.update(land);
    });
}
```

4.3. Permissions for the Network

The file to limit permissions is not given from scratch in the same way as the model file and the transaction logic file. We can create such a file and name it `permissions.acl` and write our desired rules for participants. But when hosting locally and using the `peerAdminCard`(discussed in the next section), we are merely simulating peers. So, the network throws error when we use such rules. So, I have not used permissions for peers in this project.

5. Deploying the Business Network

Once the modelling and transaction logic is finished, we can deploy the business network locally to generate a REST API. On this REST API, we can build any website required and it runs with Blockchain Database underneath. To generate the REST API, we first need to create a special archive file of defined business network. This is called Business Archive file(.bna file). To create such an archive file, execute the following command

```
$ composer archive create -t dir -n .
```

Now, we can see a `⟨business_network_name⟩@0.0.1.bna`. This archive file can now be deployed into the Hyperledger Fabric and the network can start its operations. But we need an administrator for the network. So, we for a new identity, `networkAdmin` for monitoring the network after the deployment. To install this, we use the command

```
$ composer network install --card PeerAdmin@hlfv1 --archiveFile
        <business_network_name>@0.0.1.bna
```

This command takes a card for being the administrator of the peers of the network and the archive file. We have to start the network and host the REST server. To start the business network, we need a `networkAdmin` card and also the `peerAdmin` card. We also need to have the archive file and provide required credentials for the network administrator. The following command takes care of doing all the things

```
$ composer network start --networkName <business_network_name>
--networkVersion 0.0.1 --networkAdmin admin --networkAdminEnrollSecret
adminpw --card PeerAdmin@hlfv1 --file networkadmin.card
```

But the networkAdmin identity is still not invoked in the network. The following command takes care of that

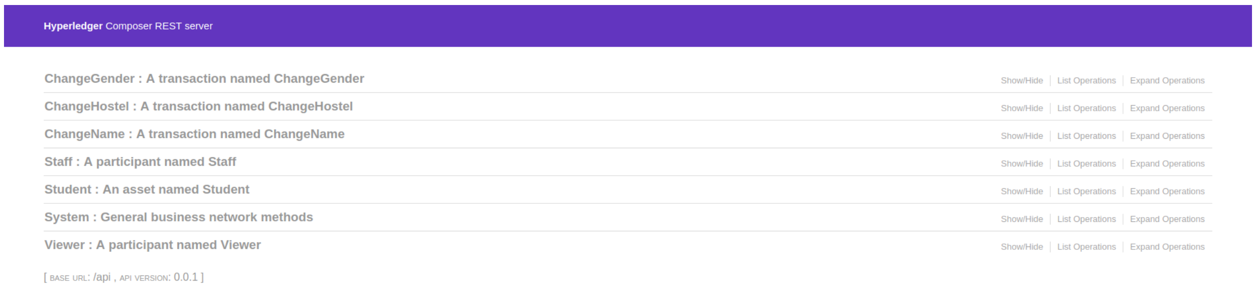
```
$ composer card import --file networkadmin.card
```

Now, we can start the REST server. Use the following command to start the REST server

```
$ composer-rest-server
```

This will give us certain options in deploying. The admin card it asks should be `admin@⟨business_network_name⟩` Next, we can allow namespaces in the generated API or not. If we allow them, all the titles will have the format, `⟨organisation_namepace⟩.⟨aspect_name⟩`. In this case, the aspect can be an asset or a participant or a transaction. For instance, `com.ac.iitm.LandTransaction`.

We have the option to secure the API but this produces some errors while viewing from other devices. It is better to leave **No** as the default in the two fields for security. We can choose to have event publication in the network or not. Now in a browser, go to `http://localhost:3000`. This will show the generated REST API. The generated API looks like Figure 1 for the students-network and respectively for land-record-network.


Hyperledger Composer REST server

ChangeGender : A transaction named ChangeGender Show/Hide | List Operations | Expand Operations
ChangeHostel : A transaction named ChangeHostel Show/Hide | List Operations | Expand Operations
ChangeName : A transaction named ChangeName Show/Hide | List Operations | Expand Operations
Staff : A participant named Staff Show/Hide | List Operations | Expand Operations
Student : An asset named Student Show/Hide | List Operations | Expand Operations
System : General business network methods Show/Hide | List Operations | Expand Operations
Viewer : A participant named Viewer Show/Hide | List Operations | Expand Operations

[BASE URL: /api , API VERSION: 0.0.1]

Figure 1: REST API generated at http://localhost:3000

6. Angular Website over REST API

After the REST server is up and running, click on any of the aspects and we can see the set of operations which can be done on that aspect. These involve basic CRUD(Create, Read, Update and Delete) operations and some with a specific identifier input. All these operations can be done passing a JSON object. A JSON object is key/value pair dictionary which support all the operations and datatypes provided by Javascript. This is the idea we are going to exploit and develop a website for easy viewing of the network from a network admin point of view. I chose angular for the project because it gives uniformity as it is developed using NodeJS and AngularJS. Both of these are Javascript APIs for console and web browser purposes respectively. Most of the modern websites such as PayPal, YouTube, the Guardian etc. are developed using AngularJS. A basic knowledge of Angular application development is required for this section. Tutorials can also be found in `https://angular.io`.

The JSON object required for assets and participants of the network is of the form

Listing 5: Example of JSON object for Assets and Participants

```
{

    $class:``<organisation_namespace>.<aspect_name>'',
    ``field_name_1'':``field_value_1'',
    ``field_name_2'':``field_value_2'',
    ...
}
```

The transactions need two new fields along with the ones defined in the model file. These are the `timestamp` of the transaction in ISO format(yyyy-mm-ddThh:mm:ssZ) and a field called `transactionId` which can be empty. But this `transactionId` is filled after the transaction takes place. So, the JSON object required for the transaction is of the form

Listing 6: Example of JSON object for Transactions

```
{

    $class:``<organisation_namespace>.<transaction_name>'',
    ``field_name_1'':``field_value_1'',
    ``field_name_2'':``field_value_2'',
    ...
    ``transactionId'':``'''',
    ``timestamp'':``yyyy-mm-ddThh:mm:ssZ''
}
```

To initiate a transaction and form a `timestamp`, we can use the Javascript helper method, `new Date().toISOString()`.

6.1. *Skeleton for the Angular Application*

Hyperledger Composer provides a way to generate the skeleton of an Angular 4 application. In the same directory as the business network, use the command

```
$ yo hyperledger-composer:angular
```

This then asks for some standard fields such as name of the website, author's name and author's email. Apart from these, the important fields that need to be filled up are the REST sever address, which is `http://localhost` and the port is `3000`. Also, we should `Connect to an existing API` and select `Namespaces not used`. After the command is executes, a new folder is created with the same name as the project's name. After navigating into this folder, use the command

```
$ npm start
```

to start the website. Then we can find the skeleton website running at `http://localhost:4200`. This uses and needs the existing REST server at `http://localhost:3000` for functioning. So, it is recommended to do host the website and the server API in separate terminal tabs or in a new terminal window.

6.2. *Improvements Done to the Angular Application*

As the Angular application developed is basic and doesn't have support to add multiple assets or even add participants and transactions. So, the following are the major developments I made to the website.

1. Ability to upload the data using .csv file.
2. Added Participants and Transactions.

There were also minor developments made such as adding an introduction to the website and changing the favicon.

6.2.1. Upload Data from .csv File

The current skeleton website doesn't have the mechanism to upload data through a file. We have to add each participant and perform each transaction by ourselves. But we could write a function in Javascript to upload a file and parse it and add the required data to the network. Here, we exploit the same idea of JSON objects. But the major problem here is that the output of such a function should be a Promise object. So, we return a Promise object after we add the data using the Javascript helper method, `toPromise()`.

The required HTML for the task will consist of a data upload field which is readily available and a `FileReader` object to parse the file. The Javascript function is written in ⟨`aspect_name`⟩`.component.ts` file which is a Javascript file with additional plugins to support server details. The Javascript code for uploading the details of the students from a .csv file goes as follows

Listing 7: Javascript function for uploading data from a .csv file

```
csvFormAdd(): void
{
      var fileToLoad = (<HTMLInputElement>document
                    .getElementById(''csvFile'')).files[0];
      var newService = this.serviceStudent;
      var fR = new FileReader();
      var objArray = [];
      fR.readAsText(fileToLoad);
      fR.onload = function(e:any)
      {
        var rows = e.target.result.split(''\n'');
        for(var i=0;i<rows.length-1;i++)
        {
          var cells = rows[i].split('','');
          var newAsset =
          {
            $class: ''com.ac.iitm.Student'',
            ''rollNo'':cells[0],
            ''name'': cells[1],
            ''roomNo'':parseInt(cells[2]),
            ''hostel'':cells[3],
            ''gender'':cells[4]
          }
          newService.addAsset(newAsset).toPromise();
          objArray.push(newAsset);
        }
      }
}
```

The working of the method is simple. We are finding the HTML element through which the file is uploaded and thus extracting the file which is uploaded. Then, we are using the `FileReader` object to parse the input file. As a .csv file is a way to organise text using commas for columns and new line character, `\n` for rows. We are splitting the input file into rows using the split method in Javascript and then correspondingly each row into cells. These cells follow a particular order i.e, roll number, name, room number etc. So, it is easy for us to form a JSON object from this and then using the network's service object to add this data to the network. This is same standard way in which we add a single element to the network. Then we are repeating the process until we exhaust the data.

6.2.2. Adding Participants and Transaction Fields to the Website

The current skeleton website even after adding the facility to upload data using a .csv file still misses some of the basic functionality. A business network consists of assets, participants and transactions. I will demonstrate the process for adding participants to the website but the process to add transactions is similar. Go to the directory, `⟨website_name⟩/src/app` and create a new folder with the name of the aspect, here the participant type. Let us say the participant currently is `Staff`. We need to create the following files in the folder

1. Staff.component.html - It is used to define the view of the page. In all pages, I have defined them in a table with two buttons to add more participants.
2. Staff.component.css - It is imported in the .ts file to give the page view styles using Bootstrap and CSS.
3. Staff.service.ts - This provides methods to interact with the REST API and these methods are extended in the .ts file. Basic CRUD operations are provided as default but changes can be made to fit the needs.
4. Staff.component.ts - Used for making the html file more active and interacts with the REST API through extended methods.

We can also find a .component.spec.ts file in assets folder but this is for testing the implementation and can be omitted. The first two files in the list form the view whereas the last two form the operational files. Then it is the matter of completing these files. I used the same format given for the assets page for these files too. The changes needed to be made were adding some functionality and deleting some unnecessary functions and changing the aspect field. Then we have to import these files in `app-routing.module.ts` and `app.module.ts` present in the main folder of the website. There were comments written previously in them to make us understand what needed to be done. For instance to make the `Staff` participant appear in the website, we need to import `StaffComponent` and `StaffService` in the file `app.module.ts` using the following statements

```
import { StaffComponent } from './Staff/Staff.component';
import { StaffService } from './Staff/Staff.service';
```

Then, add a path to these fields in the file `app-routing.module.ts` after importing the component using the statement

```
{ path: 'Staff', component: StaffComponent}
```

Now, we can add the field `Staff` in `app.component.html` and from there, we can see the web page working.

7. Problems Faced and Future Work Possible

One problem that I face is the amount of data was too much for my local storage. The students data that I used contains information about 494 students and when I uploaded it using the .csv method, initially only 46 records were showing. Then I waited and the list was growing gradually. So, if we have a designated server like the students server, we can launch the code in the server and test the full potential of the network. One thing we can notice is that as the data grew, the time taken for an additional block that can be added also grew. This is also called the Bitcoin scalability problem. It is an unsolved problem in Bitcoin network too. But we can prevent the time taken by the network using some alternate blockchain patterns such as [8] or [9]. We can also use an enterprise solution such as BigChainDB to act as a data storage for our blockchain network.

Some of the future work which can be possible is to optimise the underlying blockchain infrastructure using Hyperledger Fabric to utilise its resources to the maximum extent for our specified data. Integration of the two networks, students network and land-record network can provide a new area to experiment upon. Adding real-world data to the land-record-network can be useful to understand the constraints of real world and useful for future developments. The existing website can also be developed to add asynchronous updates using AJAX and single button queries to its features.

8. Summary of the Project

In summary, we were trying to find a way to harness the power of Blockchains as a private ledger system. This can be used in many public services such as Aadhar data keeping, records of police data etc. We experimented with different kinds of open source solutions as a part of project such as MultiChain, Hyperledger etc. We thought Hyperledger will suit the project well due to its massive documentation and online support and its convenient mechanisms. I extracted data of undergraduate students from the institute students server with the help of the Institute Web and Mobile Operations team. Then, I created two business networks, students-network and land-record-network which deal with the data of institute students and virtual land data. I created a REST API and an Angular Application for easy viewing of the network.

9. References

[1] CS6530 - *Applied Crytpography.* IIT Madras, taught by Dr. Chester Rebeiro, Jan-May 2017.

[2] CS6111 - *Foundations of Cryptography.* IIT Madras, taught by Dr. Shweta Agrawal, Jul-Nov 2017.

[3] Hyperledger Composer *Official website for Hyperledger Composer.* `https://hyperledger.github.io/composer/latest/`

[4] Hyperledger Composer Documentation *Official documentation for Hyperledger Composer.* `https://hyperledger.github.io/composer/unstable/jsdoc/`

[5] YouTube Channel *Zach Gollwitzer.* `https://www.youtube.com/channel/UCDwIw3MiPJXu5SavbZ3_a2A/`

[6] MOOC on Coursera *IBM Blockchain Foundation for Developers* `https://www.coursera.org/learn/ibm-blockchain-essentials-for-developers`

[7] MOOC on edX *Blockchain for Business - An Introduction to Hyperledger Technologies* `https://www.edx.org/course/blockchain-business-introduction-linuxfoundationx-lfs171x`

[8] Rafel Pass, Elaine Shi. *FruitChains: A Fair Blockchain.* Cornell University Press, United States of America, 2017.

[9] Micali et.al. *Algorand: Scaling Byzantine Agreements for Cryptocurrencies.* CSAIL MIT, United States of America, 2016.