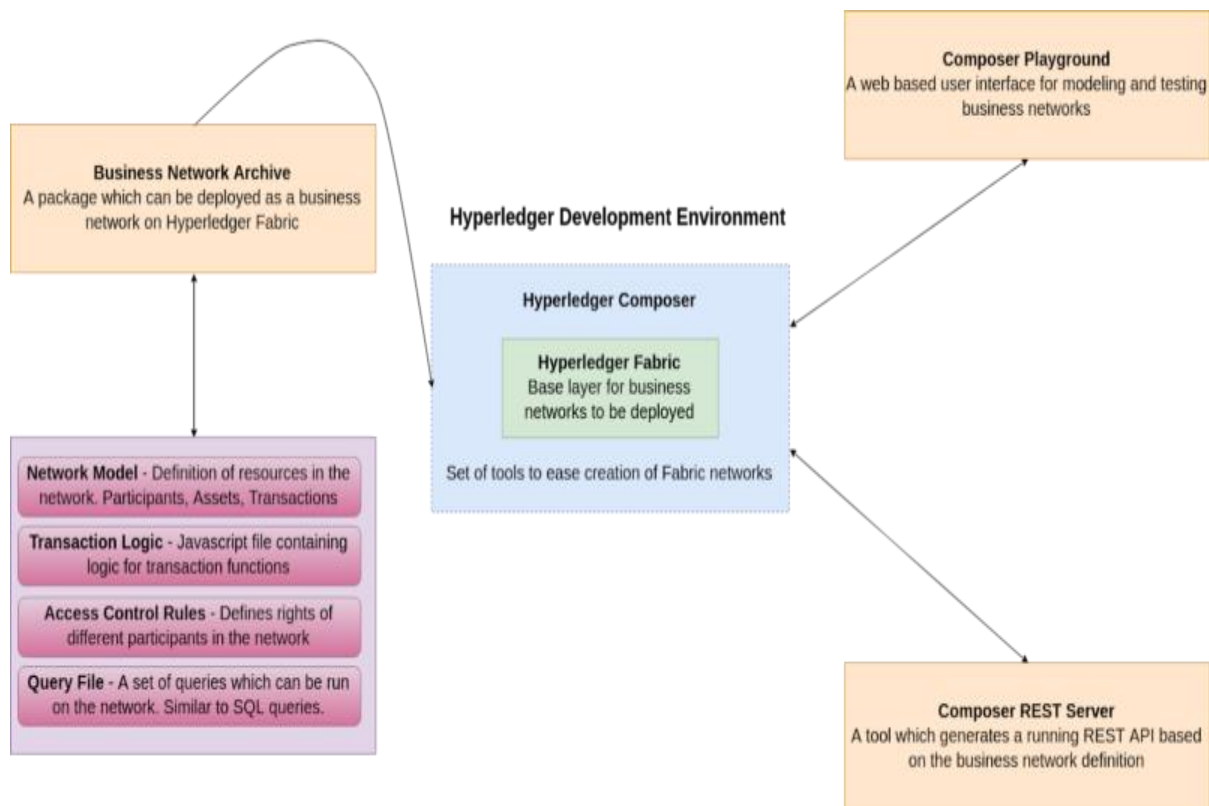


## Assignment No. C6

- **Title:** Write a program to create a Business Network using Hyperledger
- **Objective:**
- **Theory :**

### Introduction to Hyperledger Fabric and related applications



Development environment overview for Hyperledger

**Hyperledger Fabric** is an open source framework for making private (permissioned) blockchain business networks, where identities and roles of members are known to other members. The network built on fabric serves as the back-end, with a client-side application front-end. SDK's are available for Nodejs and Java to build client applications, with Python and Golang support coming soon.

**Hyperledger Composer** is a set of Javascript based tools and scripts which simplify the creation of Hyperledger Fabric networks. Using these tools, we can generate a **business network archive (BNA)** for our network. Composer broadly covers these components:

- Business Network Archive (BNA)

- Composer Playground
- Composer REST Server

**Business Network Archive** — Composer allows us to package a few different files and generate an archive which can then be deployed onto a Fabric network. To generate this archive, we need:

- **Network Model** — A definition of the resources present in the network. These resources include Assets, Participants, and Transactions. We will come back to these later.
- **Business Logic** — Logic for the transaction functions
- **Access Control Limitations** — Contains various rules which define the rights of different participants in the network. This includes, but is not limited to, defining what Assets the Participants can control.
- **Query File (optional)** — A set of queries which can be run on the network. These can be thought of as similar to SQL queries. You can read more on queries [here](#).

**Composer Playground** is a web based user interface that we can use to model and test our business network. Playground is good for modelling simple Proofs of Concept, as it uses the browser's local storage to simulate the blockchain network. However, if we are running a local Fabric runtime and have deployed a network to it, we can also access that using Playground. In this case, Playground isn't simulating the network, it's communicating with the local Fabric runtime directly.

**Composer REST Server** is a tool which allows us to generate a REST API server based on our business network definition. This API can be used by client applications and allows us to integrate non-blockchain applications in the network.

## Installing the prerequisites, tools, and a Fabric runtime

### *1. Installing Prereqs*

Now that we have a high level understanding of what is needed to build these networks, we can start developing. Before we do that, though, we need to make sure we have the prerequisites installed on our system. An updated list can be found [here](#).

- Docker Engine and Docker Compose
- Nodejs and NPM
- Git
- Python 2.7.x

For Ubuntu users, Hyperledger has a bash script available to make this process extremely easy. Run the following commands in your terminal:

Unfortunately, Mac users have to manually install the aforementioned tools and make sure they have all the prerequisites on their system. [This page](#) is kept up to date with installation instructions.

## 2. Installing tools to ease development

Run the following commands in your Terminal, and make sure you're **NOT** using sudo when running npm commands.

**composer-cli** is the only essential package. The rest aren't core components but will turn out to be extremely useful over time. We will learn more about what each of these do as we come across them.

## 3. Installing a local Hyperledger Fabric runtime

Let's go through the commands and see what they mean. First, we make and enter a new directory. Then, we download and extract the tools required to install Hyperledger Fabric.

We then specify the version of Fabric we want, at the time of writing we need 1.2, hence **hlfv12**. Then, we download the fabric runtime and start it up.

Finally, we generate a `PeerAdmin` card. Participants in a Fabric network can have business network cards, analogous to real life business cards. As we mentioned before, Fabric is a base layer for private blockchains to build upon. The holder of the `PeerAdmin` business card has the authority to deploy, delete, and manage business networks on this Fabric runtime (aka YOU!)

If everything went well, you should see an output like this:

```
The following Business Network Cards are available:
Connection Profile: hlfv1


| Card Name       | UserId    | Business Network |
|-----------------|-----------|------------------|
| PeerAdmin@hlfv1 | PeerAdmin |                  |


Issue composer card list --card <Card Name> to get details a specific card
Command succeeded
```

Also, if you type `ls` you'll see this:

```
createComposerProfile.sh  fabric-scripts  stopFabric.sh
createPeerAdminCard.sh  _loader.sh     teardownAllDocker.sh
downloadFabric.sh        package.json    teardownFabric.sh
fabric-dev-servers.tar.gz startFabric.sh
```

Basically what we did here was just download and start a local Fabric network. We can stop is using `./stopFabric.sh` if we want to. At the end of our development session, we should run `./teardownFabric.sh`

**NOTE:** This local runtime is meant to be frequently started, stopped, and torn down for development use. For a runtime with more persistent state, you'll want to deploy the network outside the dev environment. You can do this by running the network on Kubernetes or on managed platforms like IBM Blockchain. Still, you should go through this tutorial first to get an idea.

## Creating and deploying our business network

Remember the packages `yo` and `generator-hyperledger-composer` we installed earlier?

`yo` provides us a generator ecosystem where generators are plugins which can be run with the `yo` command. This is used to set up boilerplate sample applications for various projects. `generator-hyperledger-composer` is the Yo generator we will be using as it contains specs to generate boilerplate business networks among other things.

### 1. Generating a business network

Open terminal in a directory of choice and type `yo hyperledger-composer`

```
haardik@haardik-XPS-15-9570:~/workspace/f2$ yo hyperledger-composer
Welcome to the Hyperledger Composer project generator
? Please select the type of project: (Use arrow keys)
> Angular
  Business Network
  LoopBack
  Model
```

You'll be greeted with something similar to the above. Select `Business Network` and name it `cards-trading-network` as shown below:

```
haardik@haardik-XPS-15-9570:~/workspace/f2$ yo hyperledger-composer
Welcome to the Hyperledger Composer project generator
? Please select the type of project: Business Network
You can run this generator using: 'yo hyperledger-composer:businessnetwork'
Welcome to the business network generator
? Business network name: cards-trading-network
? Description: A Hyperledger Fabric network to trade cards between permissioned participants
? Author name: Haardik Haardik
? Author email: haardikk21@gmail.com
? License: Apache-2.0
? Namespace: org.example.biznet
? Do you want to generate an empty template network? Yes: generate an empty template network
  create package.json
  create README.md
  create models/org.example.biznet.cto
  create permissions.acl
  create .eslintrc.yml
haardik@haardik-XPS-15-9570:~/workspace/f2$
```

### 2. Modeling our business network

The first and most important step towards making a business network is identifying the resources present. We have four resource types in the modeling language:

- Assets
- Participants
- Transactions
- Events

For our `cards-trading-network`, we will define an asset type `TradingCard`, a participant type `Trader`, a transaction `TradeCard` and an event `TradeNotification`.

Go ahead and open the generated files in a code editor of choice. Open up `org.example.biznet.cto` which is the modeling file. Delete all the code present in it as we're gonna rewrite it (except for the namespace declaration).

This contains the specification for our asset `TradingCard`. All assets and participants need to have a unique identifier for them which we specify in the code, and in our case, it's `cardId`

Also, our asset has a `GameType cardType` property which is based off the enumerator defined below. Enums are used to specify a type which can have up to N possible values, but nothing else. In our example, no `TradingCard` can have a `cardType` other than `Baseball`, `Football`, or `Cricket`

Now, to specify our `Trader` participant resource type, add the following code in the modeling file

This is relatively simpler and quite easy to understand. We have a participant type `Trader` and they're uniquely identified by their `traderIds`.

Now, we need to add a reference to our `TradingCards` to have a reference pointing to their owner so we know who the card belongs to. To do this, add the following line inside your `TradingCard` asset:

```
--> Trader owner
```

so that the code looks like this:

This is the first time we've used `-->` and you must be wondering what this is. This is a relationship pointer. `o` and `-->` are how we differentiate between a resource's own properties vs a relationship to another resource type. Since the owner is a `Trader` which is a participant in the network, we want a reference to that `Trader` directly, and that's exactly what `-->` does.

Finally, go ahead and add this code in the modeling file which specifies what parameters will be required to make a transaction and emitting an event.

### *3. Adding logic for our transactions*

To add logic behind the `TradeCard` function, we need a Javascript logic file. Create a new directory named `lib` in your project's folder and create a new file named `logic.js` with the following code:

**NOTE:** The decorator in the comments above the function is very important. Without the `@param {org.example.biznet.TradingCard} trade`, the function has no idea which `Transaction` the code refers to from the modeling language. Also, make sure the parameter name being passed (i.e. `trade`) is the one you're passing along in the function definition right after.

This code basically checks if the specified card has `forTrade == true` and updates the card's owner in that case. Then, it fires off the `TradeNotification` event for that card.

#### 4. Defining permissions and access rules

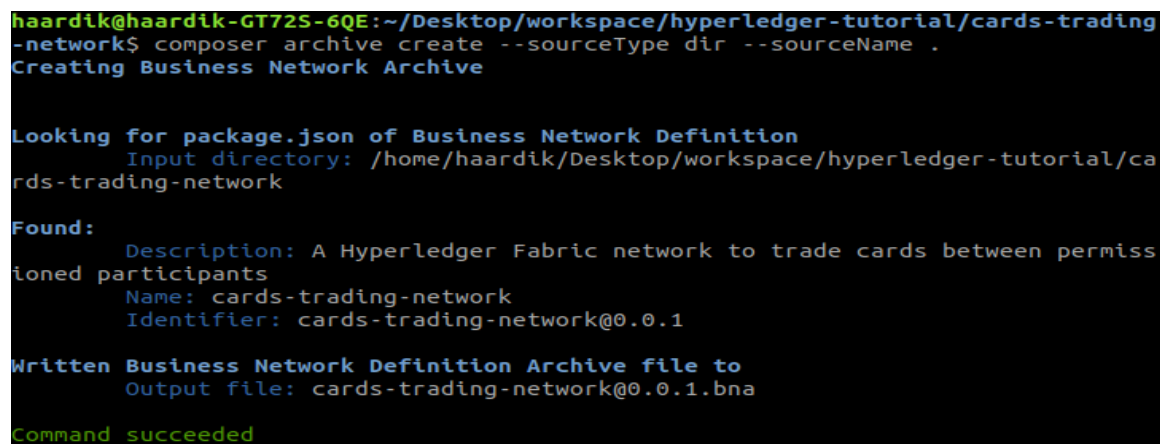
Add a new rule in `permissions.acl` to give participants access to their resources. In production, you would want to be more strict with these access rules. You can read more about them [here](#).

#### 5. Generating a Business Network Archive (BNA)

Now that all the coding is done, it's time to make an archive file for our business network so we can deploy it on our local Fabric runtime. To do this, open Terminal in your project directory and type this:

```
composer archive create --sourceType dir --sourceName .
```

This command tells Hyperledger Composer we want to build a BNA from a directory which is our current root folder.



```
haardik@haardik-GT72S-6QE:~/Desktop/workspace/hyperledger-tutorial/cards-trading-network$ composer archive create --sourceType dir --sourceName .
Creating Business Network Archive

Looking for package.json of Business Network Definition
  Input directory: /home/haardik/Desktop/workspace/hyperledger-tutorial/cards-trading-network
Found:
  Description: A Hyperledger Fabric network to trade cards between permissioned participants
  Name: cards-trading-network
  Identifier: cards-trading-network@0.0.1

Written Business Network Definition Archive file to
  Output file: cards-trading-network@0.0.1.bna

Command succeeded
```

**NOTE:** The BNA name and version come from the `package.json` file. When you add more code, you should change the version number there to deploy unique archives capable of upgrading existing business networks.

#### 6. Install and Deploy the BNA file

We can install and deploy the network to our local Fabric runtime using the `PeerAdmin` user. To install the business network, type



```
composer network install --archiveFile cards-trading-network@0.0.1.bna --card  
PeerAdmin@hlfv1
```

```
haardik@haardik-GT72S-6QE:~/Desktop/workspace/hyperledger-tutorial/cards-trading  
-network$ composer network install --archiveFile cards-trading-network@0.0.1.bna  
--card PeerAdmin@hlfv1  
✓ Installing business network. This may take a minute...  
Successfully installed business network cards-trading-network, version 0.0.1  
  
Command succeeded
```

To deploy the business network, type

```
composer network start --networkName cards-trading-network --networkVersion 0.0.1 --  
-networkAdmin admin --networkAdminEnrollSecret adminpw --card PeerAdmin@hlfv1 --  
file cards-trading-admin.card
```

```
haardik@haardik-GT72S-6QE:~/Desktop/workspace/hyperledger-tutorial/cards-trading  
-network$ composer network start --networkName cards-trading-network --networkVe  
rsion 0.0.1 --networkAdmin admin --networkAdminEnrollSecret adminpw --card PeerA  
dmin@hlfv1 --file cards-trading-admin.card  
Starting business network cards-trading-network at version 0.0.1  
  
Processing these Network Admins:  
  userName: admin  
  
✓ Starting business network definition. This may take a minute...  
Successfully created business network card:  
  Filename: cards-trading-admin.card  
  
Command succeeded
```

The `networkName` and `networkVersion` must be the same as specified in your `package.json` otherwise it won't work.

`--file` takes the name of the file to be created for THIS network's business card. This card then needs to be imported to be usable by typing

```
composer card import --file cards-trading-admin.card
```

```
haardik@haardik-GT72S-6QE:~/Desktop/workspace/hyperledger-tutorial/cards-trading  
-network$ composer card import --file cards-trading-admin.card  
  
Successfully imported business network card  
  Card file: cards-trading-admin.card  
  Card name: admin@cards-trading-network  
  
Command succeeded
```

Amazing. We can now confirm that our network is up and running by typing

```
composer network ping --card admin@cards-trading-network
```

--card this time takes the admin card of the network we want to ping.

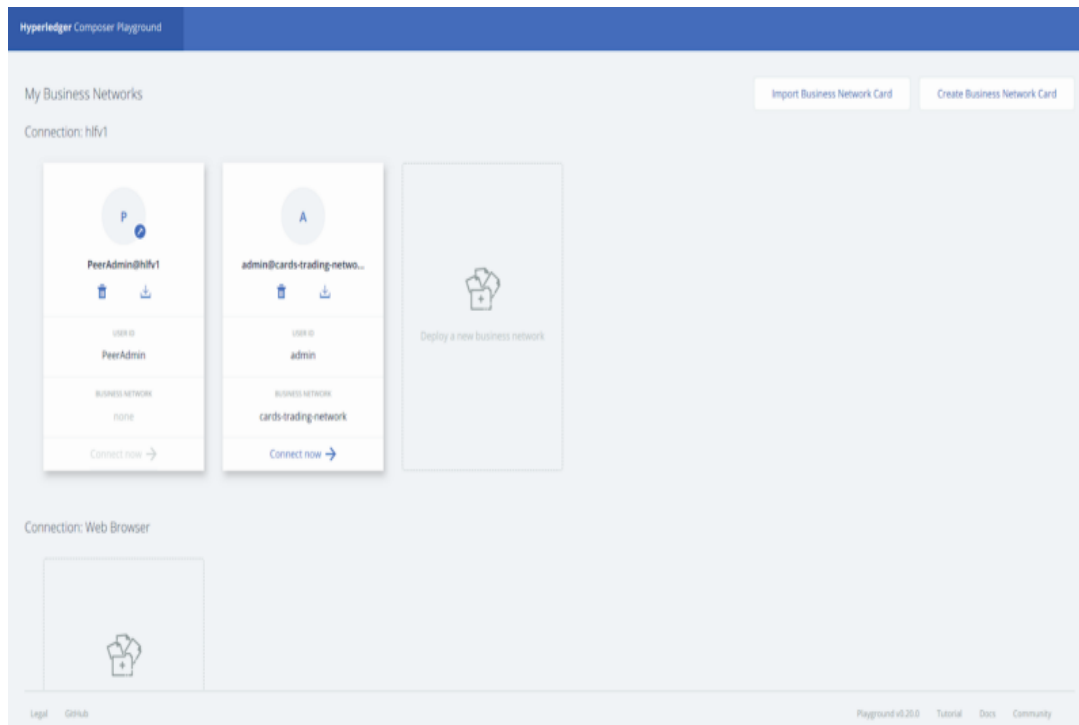
If everything went well, you should see something similar to this:

```
haardik@haardik-GT72S-6QE:~$ composer network ping --card admin@cards-trading-network
The connection to the network was successfully tested: cards-trading-network
Business network version: 0.0.4-deploy.0
Composer runtime version: 0.20.0
participant: org.hyperledger.composer.system.NetworkAdmin#admin
identity: org.hyperledger.composer.system.Identity#457abd6d405ce1dc509a16363611bae608b9904958786ae4ad5a46eb009ef10c
Command succeeded
```

Your network version will be 0.0.1 or whatever your package.json specifies — I actually forgot to take this screenshot and uploaded it after I was done writing the tutorial and making edits

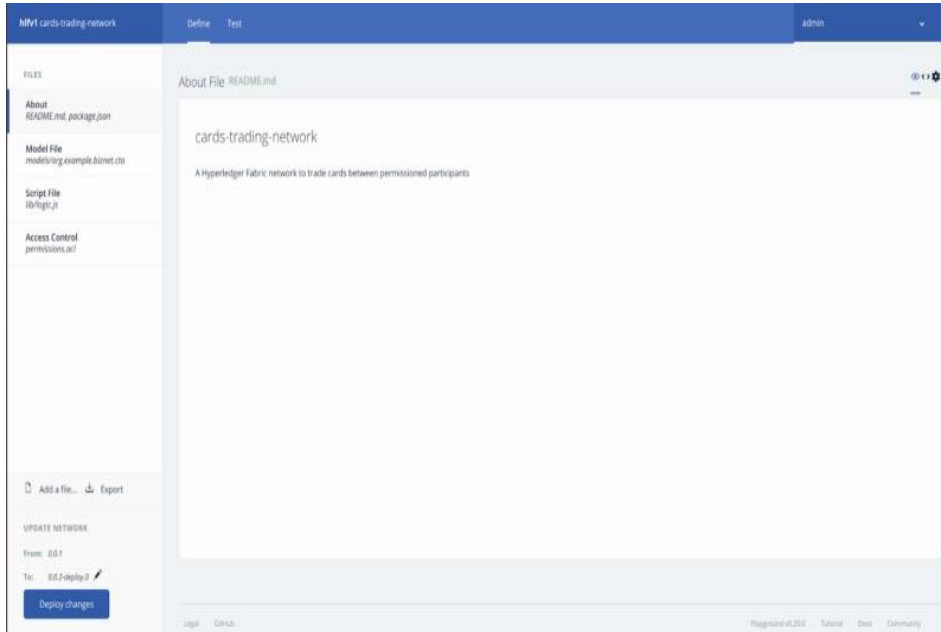
## Testing our Business Network

Now that our network is up and running on Fabric, we can start Composer Playground to interact with it. To do this, type `composer-playground` in Terminal and open up <http://localhost:8080/> in your browser and you should see something similar to this:



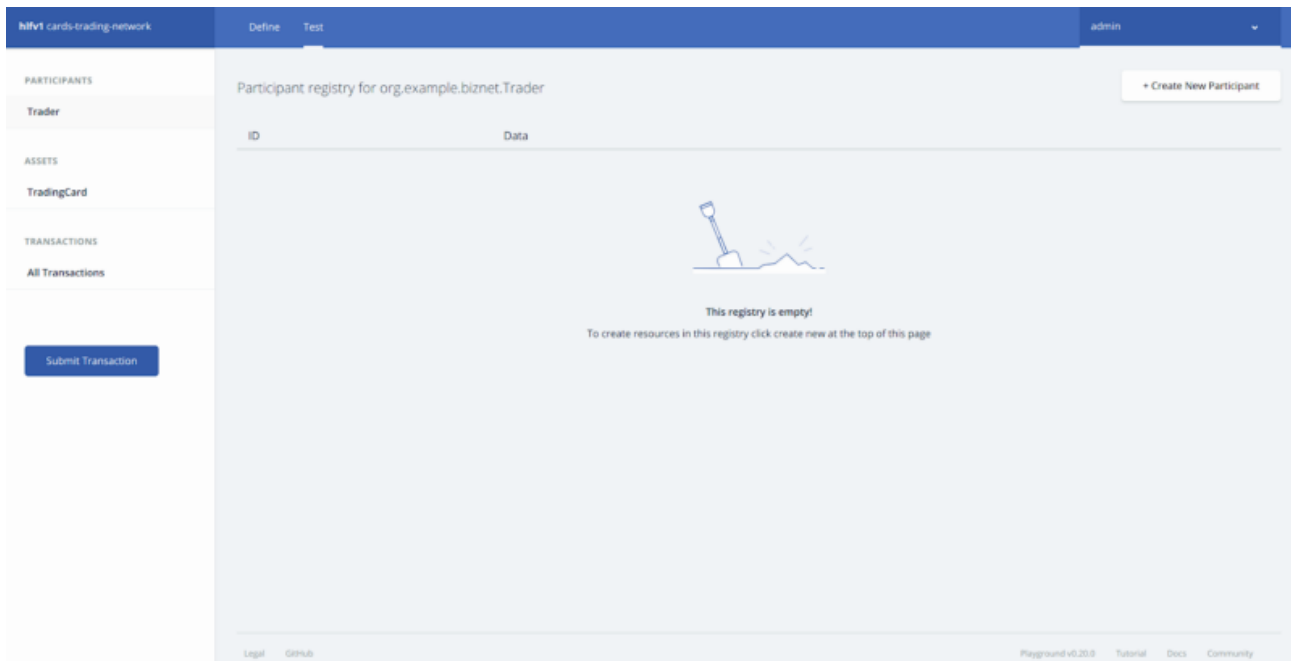
Press Connect Now for `admin@cards-trading-network` and you'll be greeted with this screen:





The **Define** page is where we can make changes to our code, deploy those changes to upgrade our network, and export business network archives.

Head over to the **Test** page from the top menu, and you'll see this:



Select **Trader** from **Participants**, click on **Create New Participant** near the top right, and make a new **Trader** similar to this:

×

Create New Participant

In registry: **org.example.biznet.Trader**

JSON Data Preview

```

1 {
2   "$class": "org.example.biznet.Trader",
3   "traderId": "1",
4   "traderName": "Haardik"
5 }

```







☐ Optional Properties

Just need quick test data? [Generate Random Data](#)

Cancel

Create New

Go ahead and make a couple more `Traders`. Here are what my three traders look like with the names Haardik, John, and Tyrone.

ID	Data	
1	<pre>{   "\$class": "org.example.biznet.Trader",   "traderId": "1",   "traderName": "Haardik" }</pre>	 
2	<pre>{   "\$class": "org.example.biznet.Trader",   "traderId": "2",   "traderName": "John" }</pre>	 
3	<pre>{   "\$class": "org.example.biznet.Trader",   "traderId": "3",   "traderName": "Tyrone" }</pre>	 

Now, let's make some Assets. Click on `TradingCard` from the left menu and press **Create New Asset**. Notice how the `owner` field is particularly interesting here, looking something like this:

```
"owner": "resource:org.example.biznet.Trader#3649"
```

This is a relationship. This is what the `--&gt;` means. We specify the exact resource type followed by their unique identifier and voila, we have a relationship pointer.

Go ahead and finish making a TradingCard something similar to this:

```
1 {
2   "$class": "org.example.biznet.TradingCard",
3   "cardId": "1",
4   "cardName": "Babe Ruth",
5   "cardDescription": "George Herman 'Babe' Ruth Jr. was an American
   professional baseball player whose career in Major League Baseball
   spanned 22 seasons, from 1914 through 1935.",
6   "cardType": "Baseball",
7   "forTrade": false,
8   "owner": "resource:org.example.biznet.Trader#1"
9 }
```

Notice how the owner fields points to Trader#1 aka Haardik for me. Go ahead and make a couple more cards, and enable a couple to have forTrade set to true.

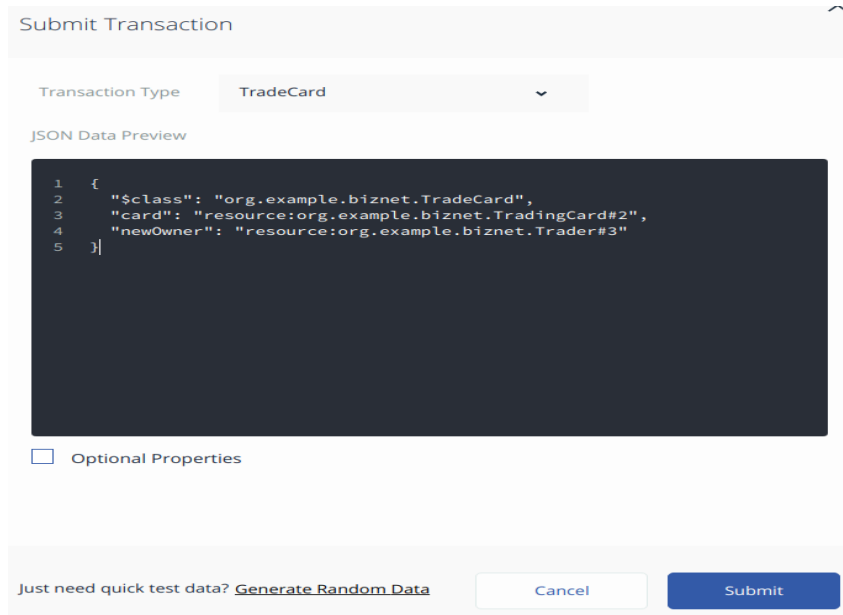
Asset registry for org.example.biznet.TradingCard

ID	Data
1	{       "\$class": "org.example.biznet.TradingCard",       "cardId": "1",       "cardName": "Babe Ruth",       "cardDescription": "George Herman 'Babe' Ruth Jr. was an Americ       "cardType": "Baseball",       "forTrade": false,       "owner": "resource:org.example.biznet.Trader#1"     }
2	{       "\$class": "org.example.biznet.TradingCard",       "cardId": "2",       "cardName": "Cy Young",       "cardDescription": "Denton True 'Cy' Young was an American Maj       "cardType": "Baseball",       "forTrade": true,       "owner": "resource:org.example.biznet.Trader#2"     }
3	{       "\$class": "org.example.biznet.TradingCard",       "cardId": "3",       "cardName": "Virat Kohli",       "cardDescription": "Virat Kohli is an Indian international cric       "cardType": "Cricket",       "forTrade": false,       "owner": "resource:org.example.biznet.Trader#3"     }

Notice how my Card#2 has forTrade == true?

Now for the fun stuff, let's try trading cards :D

Click on **Submit Transaction** in the left and make `card` point to `TradingCard#2` and `newOwner` point to `Trader#3` like this:



Submit Transaction

Transaction Type TradeCard

JSON Data Preview

```
1 {  
2   "$class": "org.example.biznet.TradeCard",  
3   "card": "resource:org.example.biznet.TradingCard#2",  
4   "newOwner": "resource:org.example.biznet.Trader#3"  
5 }
```

☐ Optional Properties

Just need quick test data? [Generate Random Data](#) Cancel Submit

Press **Submit** and take a look at your `TradingCards`, you'll see that `Card#2` now has owner `Trader#3` :D

## Generating a REST API Server

Doing transactions with Playground is nice, but not optimal. We have to make client-side software for users to provide them a seamless experience, they don't even have to necessarily know about the underlying blockchain technology. To do so, we need a better way of interacting with our business network. Thankfully, we have the `composer-rest-server` module to help us with just that.

Type `composer-rest-server` in your terminal, specify `admin@cards-trading-network`, select **never use namespaces**, and continue with the default options for the rest as follows:

```
haardik@haardik-GT725-6QE:~/Desktop/workspace/hyperledger-tutorial/cards-trading-network$ composer-rest-server
? Enter the name of the business network card to use: admin@cards-trading-network
? Specify if you want namespaces in the generated REST API: never use namespaces
? Specify if you want to use an API key to secure the REST API: No
? Specify if you want to enable authentication for the REST API using Passport: No
? Specify if you want to enable event publication over WebSockets: Yes
? Specify if you want to enable TLS security for the REST API: No

To restart the REST server using the same options, issue the following command:
  composer-rest-server -c admin@cards-trading-network -n never -w true

Discovering types from business network definition ...
Discovering the Returning Transactions..
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Generated schemas for all types in business network definition
Adding schemas for all types to Loopback ...
Added schemas for all types to Loopback
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

Open <http://localhost:3000/explorer/> and you'll be greeted with a documented version of an automatically generated REST API :D

## Generating an Angular application which uses the REST API

Remember the `yo hyperledger-composer` generator? It can do more than generating a business network. It can also create an Angular 4 application running against the REST API we created above.

To create the Angular web application, type `yo hyperledger-composer` in your Terminal, select Angular, choose to connect to an existing business network with the card `admin@cards-trading-network`, and connect to an existing REST API as well. (**Edit:** Newer versions of the software may ask for the card file instead of just the name of the card)

```
haardik@haardik-GT725-6QE:~/Desktop/workspace/hyperledger-tutorial/cards-trading-network$ yo hyperledger-composer
Welcome to the Hyperledger Composer project generator
? Please select the type of project: Angular
You can run this generator using: 'yo hyperledger-composer:angular'
Welcome to the Hyperledger Composer Angular project generator
? Do you want to connect to a running Business Network? Yes
? Project name: cards-trading-angular-app
? Description: Hyperledger Composer Angular project
? Author name: Haardik
? Author email: hhaardik@edu.uwaterloo.ca
? License: Apache-2.0
? Name of the Business Network card: admin@cards-trading-network
? Do you want to generate a new REST API or connect to an existing REST API? Connect to an existing REST API
? REST server address: http://localhost
? REST server port: 3000
? Should namespaces be used in the generated REST API? Namespaces are not used
Created application!
Completed generation process
```

This will go on to run `npm install`, give it a minute, and once it's all done you'll be able to load up <http://localhost:4200/> and be greeted with a page similar to this:

**Edit:** Newer versions of the software may require you to run `npm install` yourself and then run `npm start`

cards-trading-angular-app

Assets Participants Transactions

TradingCard [+ Create Asset](#)

cardId	cardName	cardDescription	cardType	forTrade	owner	Actions
1	Babe Ruth	George Herman "Babe" Ruth Jr. was an American professional baseball ...	Baseball	false	resource.org.example.horset.Trader1	<a href="#">Edit</a> <a href="#">Delete</a>
2	Cy Young	Denton True "Cy" Young was an American Major League Baseball pitcher ...	Baseball	true	resource.org.example.horset.Trader2	<a href="#">Edit</a> <a href="#">Delete</a>
3	Virat Kohli	Virat Kohli is an Indian international cricketer who currently captains the ...	Cricket	false	resource.org.example.horset.Trader3	<a href="#">Edit</a> <a href="#">Delete</a>

You can now play with your network from this application directly, which communicates with the network through the REST server running on port 3000.

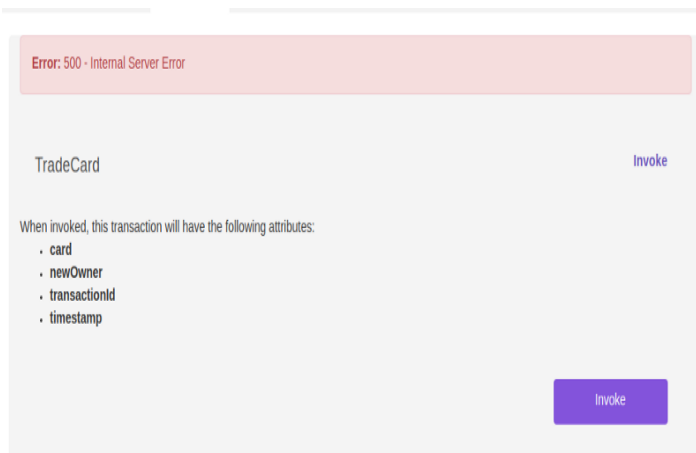
Congratulations! You just set up your first blockchain business network using Hyperledger Fabric and Hyperledger Composer :D

You can add more features to the cards trading network, setting prices on the cards and giving a balance to all `Trader`. You can also have more transactions which allow the `Traders` to toggle the value of `forTrade`. You can integrate this with non blockchain applications and allow users to buy new cards which get added to their account, which they can then further trade on the network.

The possibilities are endless, what will you make of them? Let me know in the comments :D

### KNOWN BUG: Does your Angular web app not handle Transactions properly?

At the time of writing, the angular generator has an issue where the purple `Invoke` button on the `Transactions` page doesn't do anything. To fix this, we need to make a few changes to the generated angular app.





### 1. Get a modal to open when you press the button

The first change we need to make is have the button open the modal window. The code already contains the required modal window, the button is just missing the `(click)` and `data-target` attributes.

To resolve this, open up `/cards-trading-angular-app/src/app/TradeCard/TradeCard.component.html`

The file name can vary based on your transaction name. If you have multiple transactions in your business network, you'll have to do this change across all the transaction resource type HTML files.

Scroll down till the very end and you shall see a `<button>` tag. Go ahead and add these two attributes to that tag:

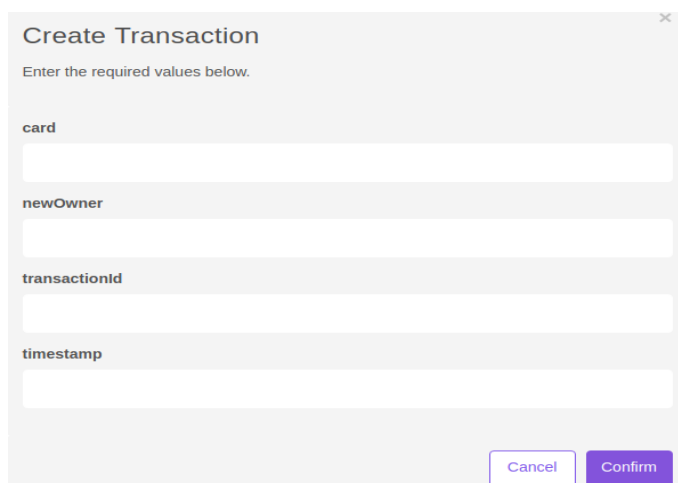
```
(click)="resetForm();" data-target="#addTransactionModal"
```

so the line looks like this:

```
<button type="button" class="btn btn-primary invokeTransactionBtn" data-  
toggle="modal" (click)="resetForm();" data-  
target="#addTransactionModal">Invoke</button>
```

The `(click)` attribute calls `resetForm()`; which sets all the input fields to empty, and `data-target` specifies the modal window to be opened upon click.

Save the file, open your browser, and try pressing the invoke button. It should open this modal:



### 2. Removing unnecessary fields

Just getting the modal to open isn't enough. We can see it requests `transactionId` and `timestamp` from us even though we didn't add those fields in our modeling file. Our network stores these values

which are intrinsic to all transactions. So, it should be able to figure out these values on it's own. And as it turns out, it actually does. These are spare fields and we can just comment them out, the REST API will handle the rest for us.

In the same file, scroll up to find the input fields and comment out the divs responsible for those input fields inside `addTransactionModal`

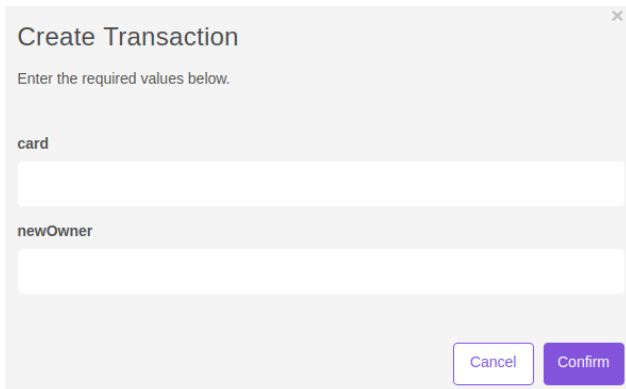
```
<!-- <div class="form-group text-left">
  <label for="transactionId">transactionId</label>

  <input formControlName="transactionId" type="text" class="form-control">
</div>

<div class="form-group text-left">
  <label for="timestamp">timestamp</label>

  <input formControlName="timestamp" type="text" class="form-control">
</div> -->
```

Save your file, open your browser, and press Invoke. You should see this:



A screenshot of a web application modal titled "Create Transaction". The modal has a close button (X) in the top right corner. Below the title, it says "Enter the required values below." There are two input fields: the first is labeled "card" and the second is labeled "newOwner". At the bottom right of the modal, there are two buttons: "Cancel" and "Confirm".

You can now create transactions here by passing data in these fields. Since `card` and `newOwner` are relationships to other resources, we can do a transaction like this:

### Create Transaction

Enter the required values below.

card

"org.example.biznet.TradingCard#2"



newOwner

"org.example.biznet.Trader#1"

Cancel

Confirm

Press **Confirm**, go back to the **Assets** page, and you will see that TradingCard#2 now belongs to Trader#1:

2	Cy Young	Denton True 'Cy' Young was an American Major Leag...	Baseball	true	resource:org.example.biznet.Trader#1	 
---	----------	--	----------	------	--------------------------------------	---