**Development of a Blockchain Application for a Food Delivery System**

Authors: Ryan Jaipersaud, Tae Hyun Koh

Instructor: Professor Anita Raja

Ph-360

Cooper Union for the Advancement of Science and Art

Date of submission 5/8/18

**Motivation and Related Work**

The food delivery market nowadays is dominated by third-party applications that pairs a customer with a restaurant and oversee their transaction. These “food deliver apps” make it convenient for customers to order and help restaurants get more exposure to customers. However, since these applications act as a middleman which has control over the transactions, they can charge whatever commission fees they want. (On average, these applications charge 13.5% commission fee on the entire order. [1]) High commission fees are profit for companies like GrubHub, but it hurts both the customer and the restaurant. This project aimed to use blockchain to minimize commission fees of food delivery services by eliminating the need for a third-party application to overlook a transaction.

A handful of blockchains applications related to food were explored already. For example, a company called “Munchie” tried using their in-app crypto currency for food delivery. Moreover, several companies like INS Ecosystems, Ripe and Foodcoin integrated blockchain to tracking agricultural produce. However, no work had been done on integrating blockchain into food delivery system. The state of the art still remains to be third party delivery applications that oversee transaction between the restaurant and the customer.

**Platform and Solution**

This project used Hyperledger as the platform to build on. Hyperledger is an open-source blockchain projects that allows developers to customize Hyperledger Fabric to create and customize a blockchain network. Hyperledger had several advantages over other existing blockchain frameworks for this project. Transactions in Hyperledger does not require mining or proof-of-work. Hence, no transaction fee is needed in a transaction between a restaurant and a customer. Moreover, it allows for the creation of different types of participants in the network and regulation of those participants’ access control to the blockchain. This made it easier for the project to have a multi-agent aspect as well as secure privacy of the participants.

The core problem that the project set to solve was ensuring a successful transaction where the customer pays when he gets the food and the restaurant gets paid when it delivers the food. The project solved the problem by creating a blockchain that integrates smart contract and cryptography.

The project involved three types of participants; Customer, Restaurant, and Public Restaurant. Restaurant and Customer are both private participants who are not visible to each other and are the participants that take part in the transaction. Public Restaurant is a public participant that that represents the linked Restaurant participant and was used to record the rating given to the restaurant after each successful transaction. The delivery starts by the Customer submitting a PlaceOrder transaction on to the network which creates a new asset in the chain. The function takes the order details (food item, amount, etc.) and assigns the owner of the asset; the Customer submitting the order and the Restaurant the order is sent to. Moreover, when the PlaceOrder transaction is passed, a key composed of a hexadecimal string for the customer is randomly generated and a cryptographic hash (using SHA-256) of that string is recorded onto the asset. Next, the restaurant confirms the order by submitting an UpdateOrderStatus transaction which changes the status of the asset and also creates a randomly generated key for the Restaurant and records the hash of that onto the food order asset. When the food is delivered to the Customer, the two parties can verify and trust the transaction by revealing each other’s randomly generated string. This is possible since the random keys specific to the order were automatically generated by the smart contract so the validity of the key can be proven by passing them through the SHA-256 hash function and compared to the hashed keys recorded on the asset. When both parties confirm the order, the rating is updated on the Public Restaurant participant helping keep a record of the Restaurant’s credibility. Although it is not implemented in the project, Customer’s payment should also be automatically sent to the Restaurant after the confirmation of the order.

The project borrowed ideas from many of the topics in class. Blockchain nature of irreversibility and transparency is the backbone of the system since it allows for the recording of data on the asset file and the recording of the credibility of the restaurant. The whole transaction is a series of smart contracts designed to eliminate the need of a third party to oversee the transaction while imposing a fraud-free exchange. Cryptographic hashing and the cryptocurrency’s (Bitcoin) use of it is what made the smart contracts work by allowing participants to digitally sign the transaction and prove their signature by digital keys.

**Team Roles:**

Ryan Jaipersaud

1. Setting up Hyperledger Fabric, REST API, Angular
2. Determining appropriate syntax to submit a successful transaction through an Ubuntu terminal
3. Developing instructions on how to setup the network

Tae Hyun Koh

1. Determining outline and pseudo code of functions in order to successfully submit a transaction to be broadcasted to the network
2. Researching the role of how blockchain can potentially be used to minimize transaction fees for food delivery platforms

Both teammates contributed equally toward the development of the make, access and javascript files that were needed to deploy the network. Both teammates contributed equally to the development of the credit system to implement multiagent system features. Both teammates were responsible for debugging code error that resulted from software relating to the network as well as error that resulted from integrating the network with Fabric and the Rest API.

**Demo and Instruction on how to setup the blockchain network**

1. Inorder to successfully deploy the business network the “teardownFabri.sh” script was excuted followed by the command “rm – Rf ~/.composer/” to clear any business network that was already being run and delete any identities on the network, respectively. The command “./startFabir.sh” was used to start up the network.As shown below in Figure 1.

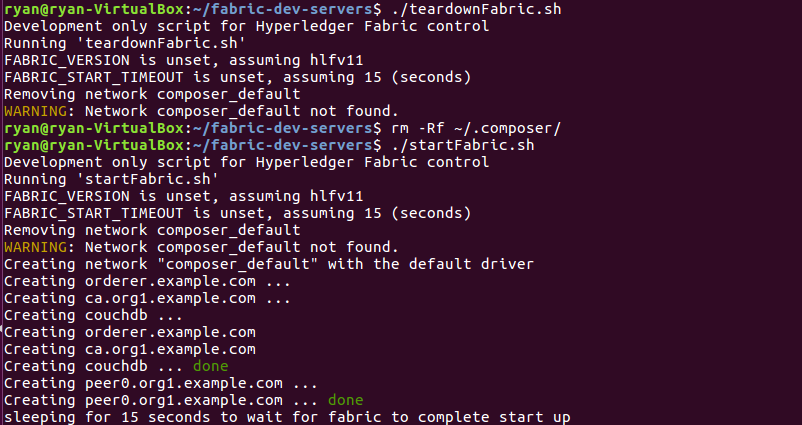


Figure 1: Resetting the network

1. The network that you want to deploy must be inside the “~/fabric-dev-servers” directory and must be extracted into a directory from that point of your choosing as shown in Figure 2.



Figure 2: directory setup

1. Once inside of the directory were you extracted the file execute the command “composer archive create –t dir –n .” to create a “.bna” file as shown in Figure 3.

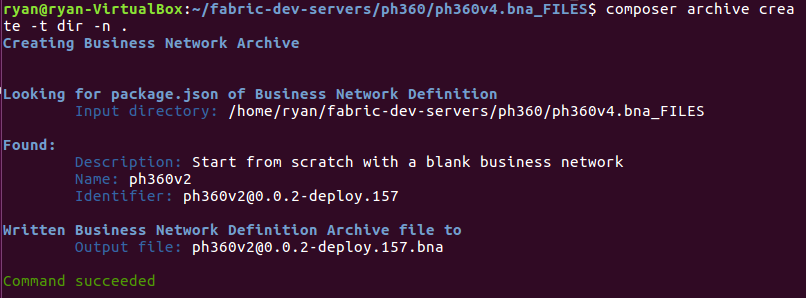


Figure 3: Creating a “.bna” file

1. Go back to the “fabric-dev-servers” directory and create a Peer Admin Card using the command “./createPeerAdminCard.sh” as shown in Figure 4.

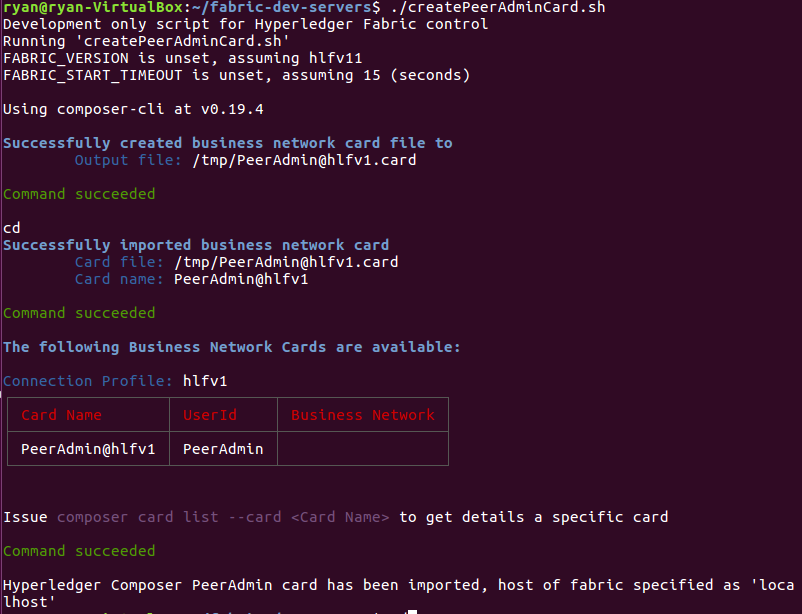


Figure 4: Creating a Peer Admin Card

1. Now a network card must be installed and the network need to be started as shown in Figure 5 back in the network directory.

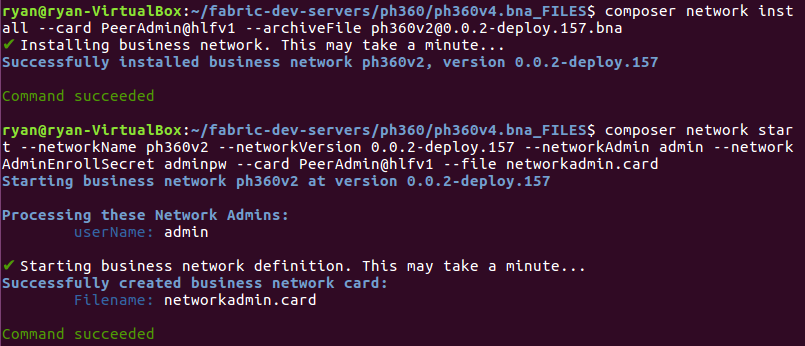


Figure 5: Install Admin Card and starting network

1. The Admin card needs to be imported and the network should be pinged to make sure it is running smoothly as shown in Figure 6.

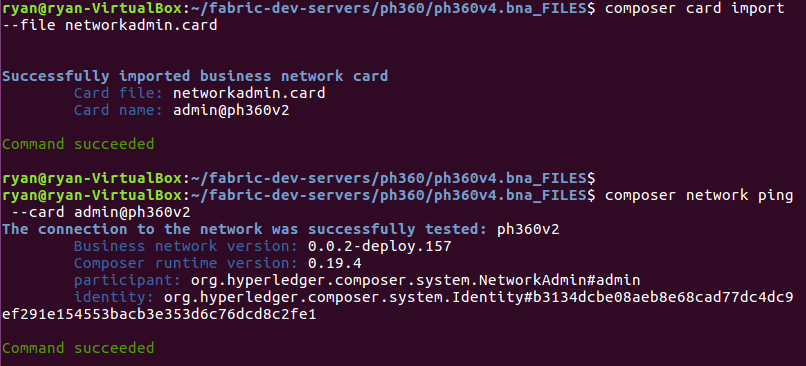


Figure 6: Card Import and Network Ping

1. Participant need to be added and identities now need to be issued to give users access to the block chain. An example of adding a Customer to a registry as well as issuing an identity is shown in Figure 7.

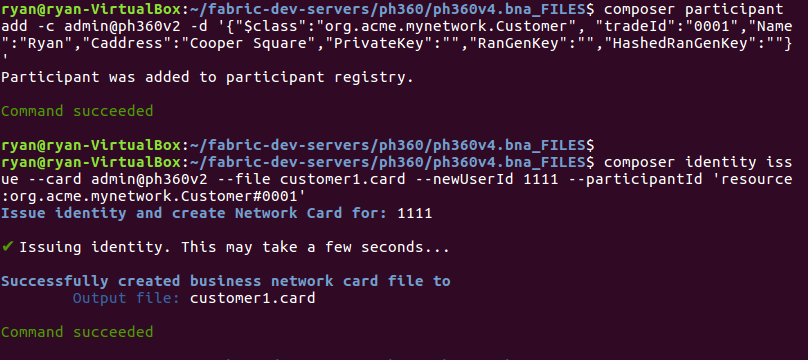
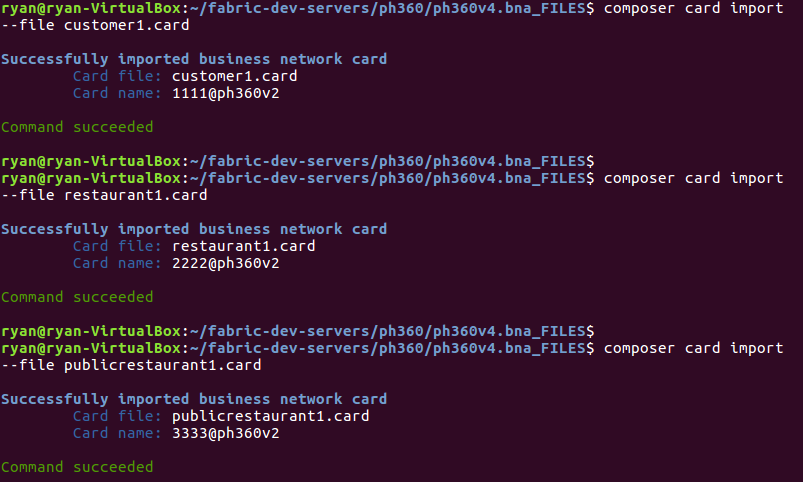


Figure 7: Adding Participant and Issuing Identities

1. For each participant that was added to a registry the corresponding card must be imported as shown in Figure 8.



1. To view the participants that were added a REST API can be generated by opening another terminal in the same directory an entering the input shown in Figure 9 with an appropriate output shown in Figure 10. The REST API will run on local port 3000.

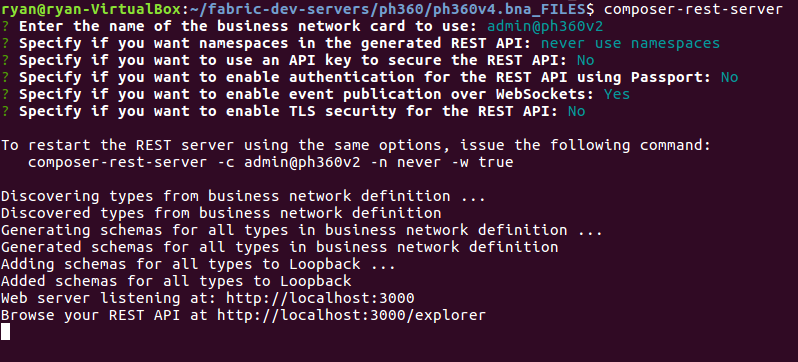


Figure 9: Generating a REST API

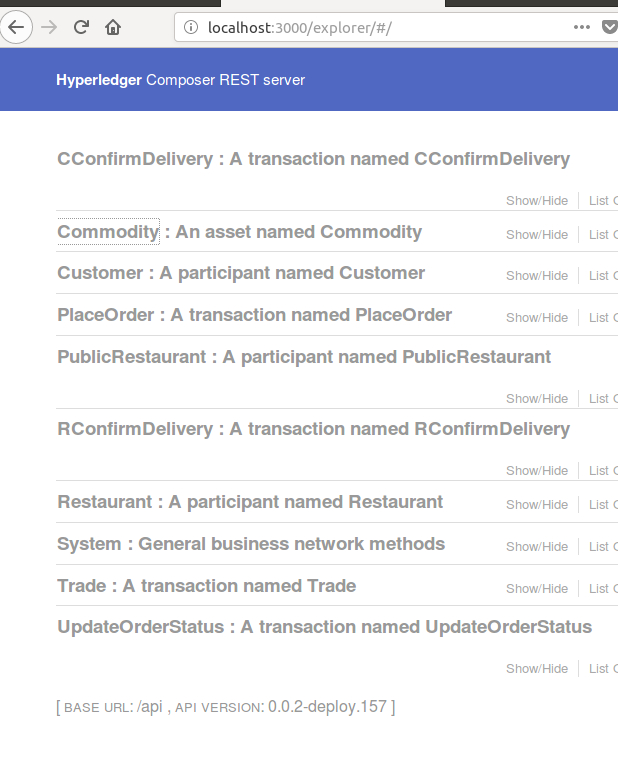


Figure 10: REST API Server

1. In order have a successful transaction between two parties the following input needs to occur as shown in Figure 11. In Figure 11 only the “Place Order” and “UpdateOrderStatus transaction are submitted. This generate key and hashes for the participant involved which can be seen by accessing the “get” function in the REST API. This also generate a commodity asset as shown in Figure 12.

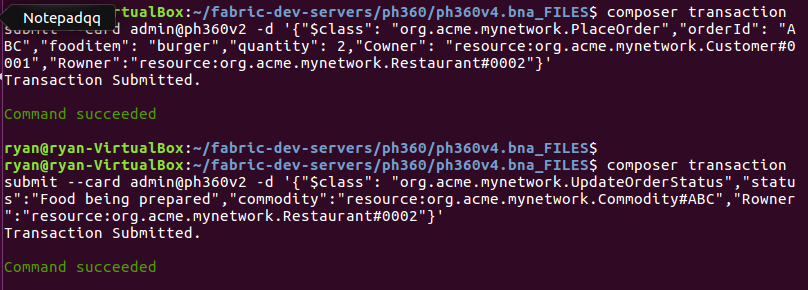


Figure 11: PlaceOrder and UpdateOrderStatus transactions

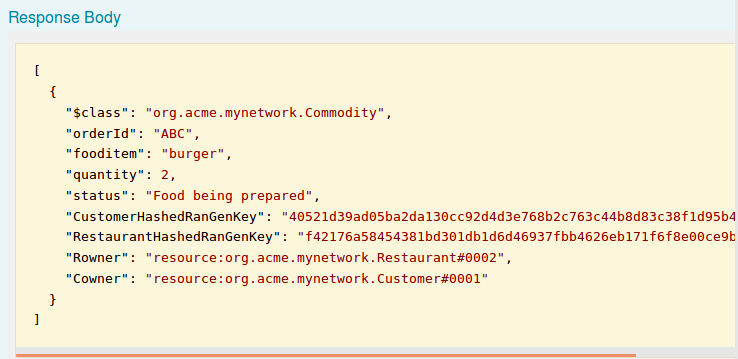


Figure 12: REST API output for a commodity

1. In order to submit the “CConfirmDelivery” and “RConfirmDelivery” the following input needs to occur. The customer and the restaurant need to give each other their respective keys so that the other can verify they are dealing with the correct party. This is the definition of a successful transaction. Appropriate output is shown in Figure 14.

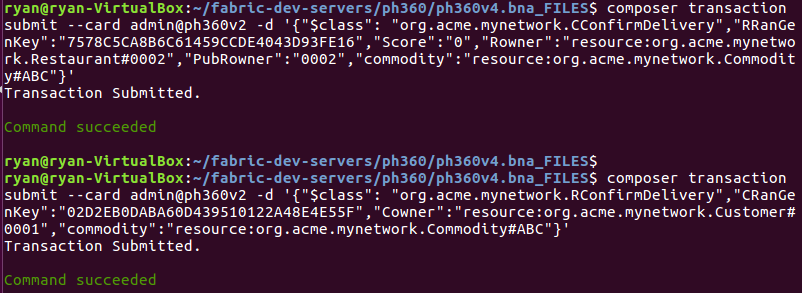


Figure 13: “CConfirmDelivery” and “RConfirmDelivery” transactions

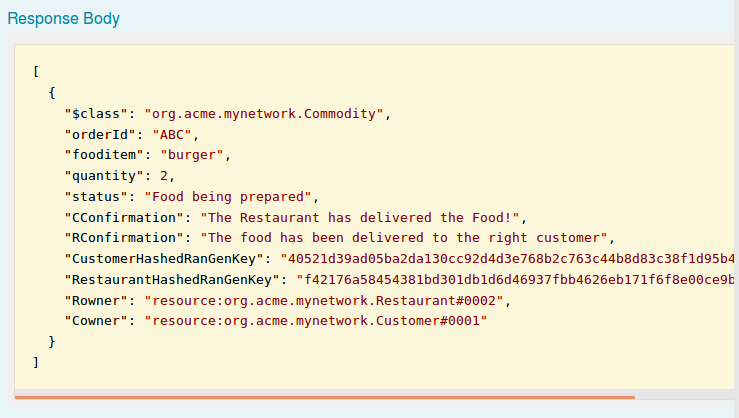


Figure 14: A successful transaction

1. To start angular one can run the command “yo hyperledger-composer:angular” as shown in Figure 15 and then go into the angular app-directory and run “npm start” . In a browser you can go to port <http://localhost:4200/> to view the angular app. An appropriate output for angular is shown in Figure 16.

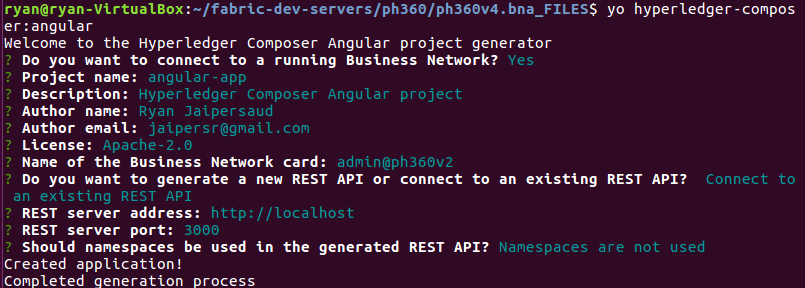


Figure 15: starting angular

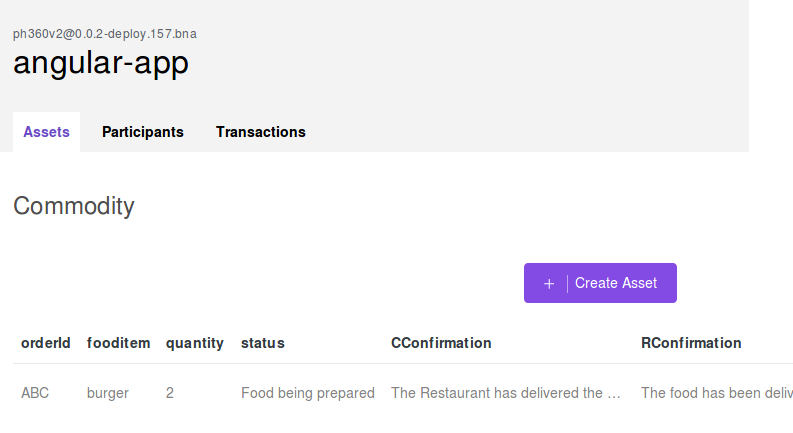


Figure 16: Angular Framework

**Project Evaluation**

The platform created showed how two parties, a restaurant and a customer, can interact such that the two can engage in a safe transaction as shown in Figure 14. Both parties were given a key and the hash of the key but only the hash was broadcasted to the network on the asset. The address of the customer was given to the restaurant without the need for other third parties to see the customers address and thus anonymity was maintained. Once the restaurant delivered the food both parties would inform the network of a successful transaction. After a successful delivery the customer had the option to give the restaurant a “Score” for quality of service. This value is publicly broadcast which shows a successful multiagent system. This platform models a real life app such as GrubHub to a fair extent.

Limitations were seen in this project. Only one computer could be connected to the network at a time. For this to be practical other people that have access to the network need to be able to broadcast transactions from other places. Another problem is the interface. As can be seen in the instructions the transaction were submitted line by line into the terminal. An appropriate user interface that allows for a minimum of 4 clicks to correspond to the 4 transactions would increase practicality.

An accepted limitation of the project is the centralized nature of issuing Ids. While blockchain is known for its decentralized nature, Hyperledger issues Ids to get on the network which is something that is accepted to minimize the transaction costs of current food delivery systems. An important feature to note is that this project makes no reference to mining as used in Bitcoin and as a result there is no need for gas.

**Project Challenges and things that were easy**

Most of the challenges associated with this project were mainly about the setup. An extensive amount of resources were needed to deploy a network. Ubuntu 16.04 was needed as well as docker, python and an appropriate version of node. If one does not have Ubuntu a virtual machine is needed and if one does not realize at least 15 GB of hard drive space is needed this could lead to wasted time. Another issue was actually creating participants and adding them to a network. The development tutorial at “https://hyperledger.github.io/composer/latest/tutorials/developer-tutorial.html” provided a good guess as where to start but lacks the syntax to actually submit transactions and as such required further research. A video detailing the perishable network tutorial helped elucidate the format of a transaction and the important of access file. The URL is as follows: “https://www.youtube.com/watch?v=FLBZu0BkppU”. Determining these resources proved to be time consuming since this is a relatively new concept. Things pertaining to Hyperledger Fabric are all over the place and there seems to be a lack of consensus on how best to approach the solution. Hyperledger Composer Playground was very easy to use and understand. We recommend developing all functionality in the playground environment and then export the “.bna” file to be deployed on Fabric. The learning curve for programming in javascript was a little time consuming but was nothing compared to the setup. Further time needs to be spent understanding the full functionality of the access file to limit access to participants.

**Project Timeline**

This project changed in a very subtle manner from the initial to final stages. Originally it was thought to communicate a successful transaction a Public Key and Private Key was needed. However, only a randomly generated key was needed for this project. This could change in the future if a currency system is built on top of the network. That is one of the main distinctions between GrubHub and this application. Not only are restaurants broadcasted on GrubHub but, Grubhub also acts as a validator of a customer’s ability to pay. The mulitagent credit system remained the same throughout the project.

Also in order to maintain anonymity and not have the Customer broadcast their address to everyone a new participant had to be created called PublicRestaurant. This is the version of the restaurant that all participants can see and as a result this is where the “Score” is viewed. By doing this the Customer can send their address to the Restaurant which will represent the same entity as PublicRestaurant where both the Customer and Restaurant Participant’s information are hidden from everyone on the network. This also hides the keys from everyone on the network.

**References**

[1] Forbes, P.; GrubHub Charges Restaurants on Average 13.5 % Commission Per Order; https://www.eater.com/2014/3/3/6270739/grubhub-charges-restaurants-an-average-13-5-commission-per-order (accessed April 11th 2018).

[2]Developer Works TV; Hyperledger Composer - Advanced Blockchain Network, https://www.youtube.com/watch?v=FLBZu0BkppU (accessed May 5th, 2017).

[3] IBM; Developer tutorial for creating a Hyperledger Composer Solution, https://hyperledger.github.io/composer/latest/tutorials/developer-tutorial.html (accessed May 5th,2017).