A PROJECT REPORT
ON

“Decentralized auction engine using Blockchain“

BY

MAYANK SENANI	MT2017067
SHARAD MAHAJAN	MT2017103
TARUN AGRAWAL	MT2017127

UNDER THE GUIDANCE OF
Prof. Shrisha Rao

Acknowledgements

We are profoundly grateful to Prof. Shrisha Rao for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

Mayank Senani
Sharad Mahajan
Tarun Agrawal

ABSTRACT

Imagine a world where business must be done between non-trusting parties, a world where there is a centralized database which is handled by one of those parties, then a centralized bookkeeping results into trust issues among the rest of the parties. To have a consistent distributed replicated ledger shared among all the non-trusting parties we use blockchain technology. Our solution is a new, open, decentralized platform that is dedicated to real time Management of any auction, worldwide, whether online or live so that it is accessible to all. In effect, everyone will be able to connect in order to sell, bid, organize an auction or offer a service to the ecosystem. our solution is **Scalable**, **reliable**, **transparent** and **Interoperable** i.e each operation will be registered within the network in a way that is transparent, publicly verifiable and infeasible to falsify.The bidder will be able to participate in numerous worldwide auctions– using just one interface.

GitHub url: https://github.com/senani-mayank/Auction-Engine

Contents

1. Introduction

Problem Statement of the project: To Develop a “Decentralized Auction Engine Using Blockchain Technology” where users can create and manager various auctions.

1.1 Motivation

Following Are key points behind Motivation for our Project :-

1. All the auction platforms that currently exist are based on one centralized operation.
2. They are not transparent i.e Bidders have no way to ensure the origin, authenticity and legitimacy of a higher bid. Only the organizer has this information.
3. They are not open i.e Every bidder must use the organizer’s platform for registration, authentication and bidding. So, as there are a huge number of auctions organizers worldwide, a bidder must register many times over. They must therefore manage many different accounts and learn how to use a new and different interface each time.
4. The ecosystem is very limited i.e Because no one standard exists, each organizer has developed its own bidder interfaces and tools and even most auction organizers don’t have mobile applications.
5. In contrast with the live auction, there will be high organizational cost and maintenance cost with limited number of bidders.

Thus, there is a need for cost effective, robust and open platform which facilitates creating and managing various auctions without any requirement of third party.

Hence, with help of underlying blockchain in the Hyperledger Fabric, we can create such platform as each participant has its own(but synchronized across network) copy of ledger(where transactions are stored) which make it “Tamper Proof” and using “Smart Contracts”,we can create auction rules which will be the business logic of our transactions and are visible to everyone and hence everyone knows what’s going in the network.

2. Technologies Used

2.1 System Requirements

Prerequisites for installing the required development tools:

- To run Hyperledger Composer and Hyperledger Fabric, at least 4Gb of memory .
- Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12
- Docker Engine: Version 17.03 or higher
- Docker-Compose: Version 1.8 or higher
- Node: 8.9 or higher (note version 9 is not supported)
- npm: v5.x
- git: 2.9.x or higher
- Python: 2.7.x
- A code editor: VSCode.

For Ubuntu, download the prerequisites using the following commands:

```
curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh



chmod u+x prereqs-ubuntu.sh
```

Next run the script - as this briefly uses sudo during its execution, you will be prompted for your password.

```
./prereqs-ubuntu.sh
```

2.2 Installing components

Step 1: Install the CLI tools

There are a few useful CLI tools for Composer developers. The most important one is `composer-cli`, which contains all the essential operations.

1. Essential CLI tools:

```
npm install -g composer-cli
```

2. Utility for running a REST Server on your machine to expose your business networks as RESTful APIs:

```
npm install -g composer-rest-server
```

3. Useful utility for generating application assets:

```
npm install -g generator-hyperledger-composer
```

4. Yeoman is a tool for generating applications, which utilises `generator-hyperledger-composer`:

```
npm install -g yo
```

Step 2: Install Playground

1. Browser app for simple editing and testing Business Networks:

```
npm install -g composer-playground
```

Step 3: Install Hyperledger Fabric

This step gives you a local Hyperledger Fabric runtime to deploy your business networks to.

1. In a directory of your choice (we will assume `~/fabric-dev-servers`), get the `.tar.gz` file that contains the tools to install Hyperledger Fabric:

```
mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers

curl -O https://raw.githubusercontent.com/hyperledger/composer-
tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz

tar -xvf fabric-dev-servers.tar.gz
```

2. Use the scripts you just downloaded and extracted to download a local Hyperledger Fabric runtime:

```
cd ~/fabric-dev-servers

./downloadFabric.sh
```

2.3 Controlling your dev environment

2.3.1 Starting and stopping Hyperledger Fabric

control your runtime using a set of scripts which you'll find in `~/fabric-dev-servers`.

The first time you start up a new runtime, you'll need to run the start script, then generate a PeerAdmin card:

```
    cd ~/fabric-dev-servers

    ./startFabric.sh

    ./createPeerAdminCard.sh
```

To stop the fabric network.

```
    ./stopFabric.sh
```

To start a new fabric connection.

```
    ./teardownFabric.sh

    ./startFabric.sh
```

2.3.2 Installing the project application-

1. To run the application, clone our github repository.

```
$ git clone https://github.com/senani-mayank/Auction-Engine

$ cd Auction-Engine
```

2. Run the commands to deploy business network on fabric.

   1. Create auction.bna file

```
cd code
composer archive create -t dir -n .
```

   2. Install network on peers

```
composer network install -a auctionnetwork@0.0.1.bna  -c PeerAdmin@hlfv1
```

   3. start Network

```
composer network start -n auctionnetwork-V 0.0.1 -A admin -S adminpw -c
PeerAdmin@hlfv1
```

   4. import card

```
  composer card import -f admin@auction_network.card
```

   5. now run rest server

```
  composer-rest-server -c admin@auction_network   -n never -w true
```

2.3.3 To run the web application :

```
$ cd code/webapp

$ http-server start -p portnumber
```

Our web application has been set to listen to the port 9990, so we need to specify this in the client browser along with IP address of the device where the application is being hosted. For example in our use case we had to specify the address of IP https://localhost:9990 for the users to connect to auction engine web application from a web browser like Google Chrome.

2.4 Front-End

We used HTML, CSS and Bootstrap and AngularJS for the web application so that the user can see and interact with the module directly.

2.5 Back-End

Back-end is developed using Node.js, composer language CTO.

3. Work-flow of Project:

3.1 Architecture of Auction-Engine

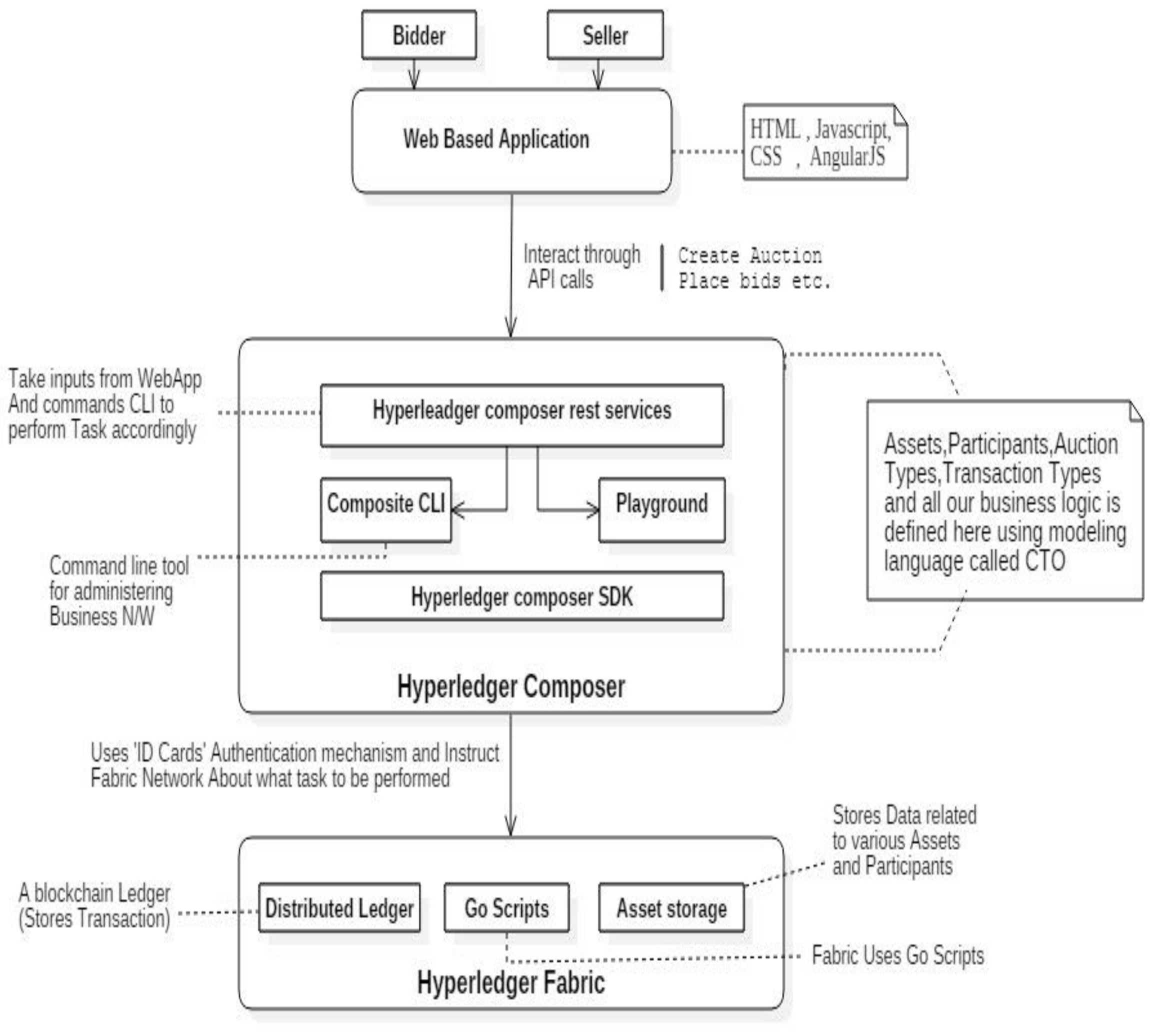

Bidder
Seller
Web Based Application
HTML , Javascript, CSS , AngularJS
Interact through API calls
Create Auction
Place bids etc.
Take inputs from WebApp And commands CLI to perform Task accordingly
Hyperleadger composer rest services
Composite CLI
Playground
Command line tool for administering Business N/W
Hyperledger composer SDK
Hyperledger Composer
Assets,Participants,Auction Types,Transaction Types and all our business logic is defined here using modeling language called CTO
Uses 'ID Cards' Authentication mechanism and Instruct Fabric Network About what task to be performed
Stores Data related to various Assets and Participants
A blockchain Ledger (Stores Transaction)
Distributed Ledger
Go Scripts
Asset storage
Fabric Uses Go Scripts
Hyperledger Fabric

Figure 3.1: Architecture of system

3.2 Flow Diagram of Auction-Engine

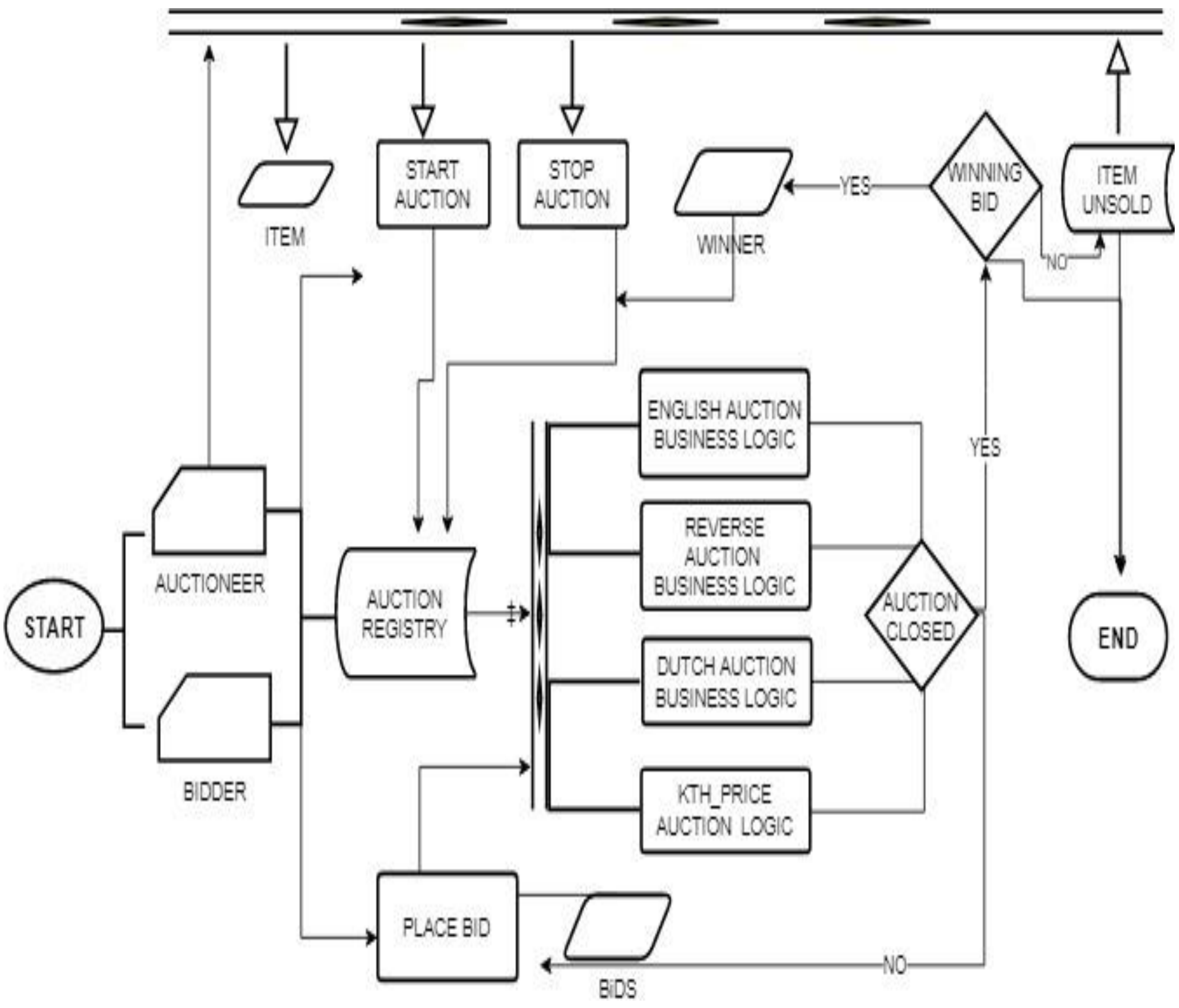


Figure 3.2: flow of auctions

3.3 Use-Case Diagram of Auction-Engine

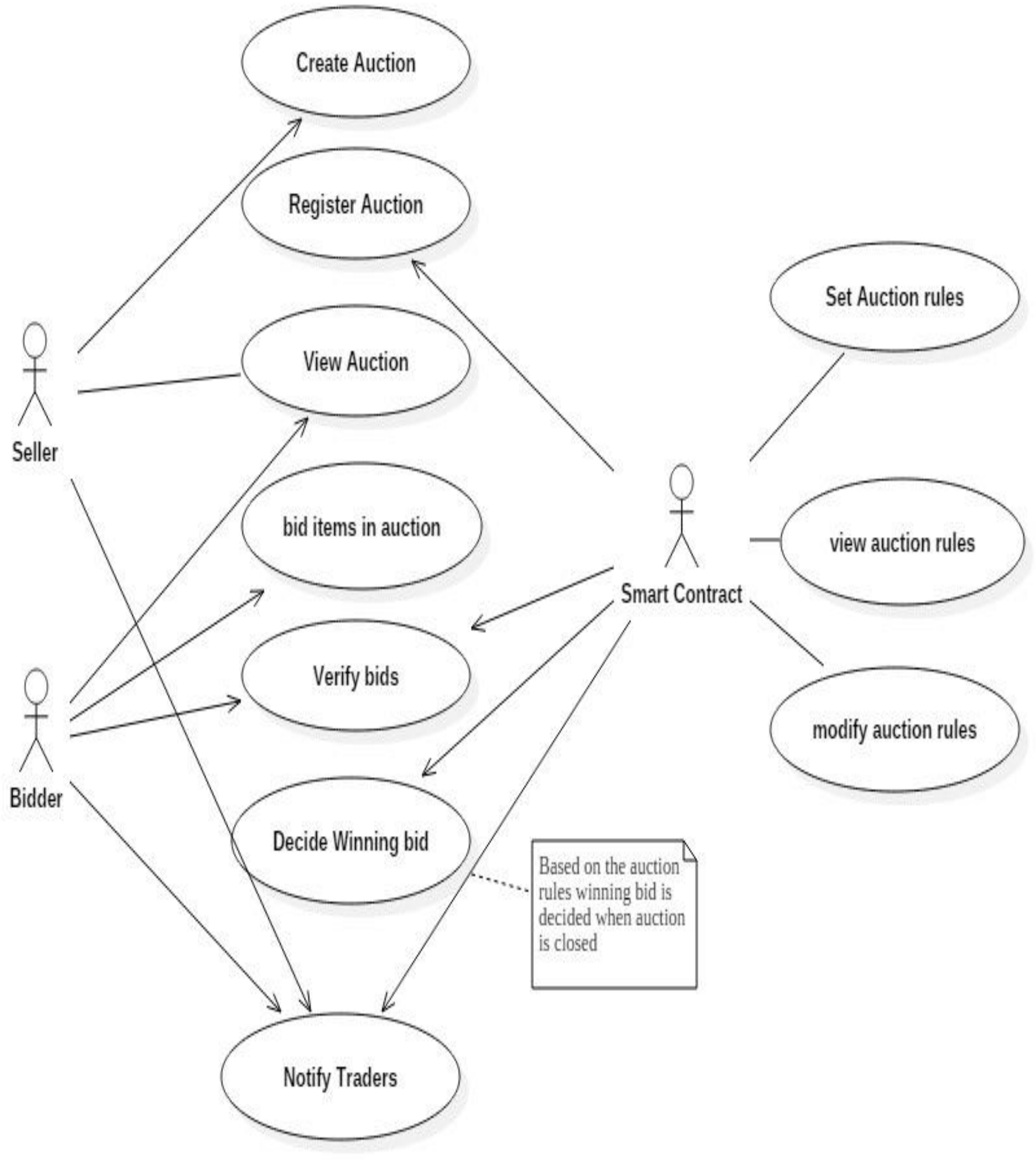


Figure 3.3: use-case of system

<u>User Roles</u> :

a user can play following two roles in our application :-

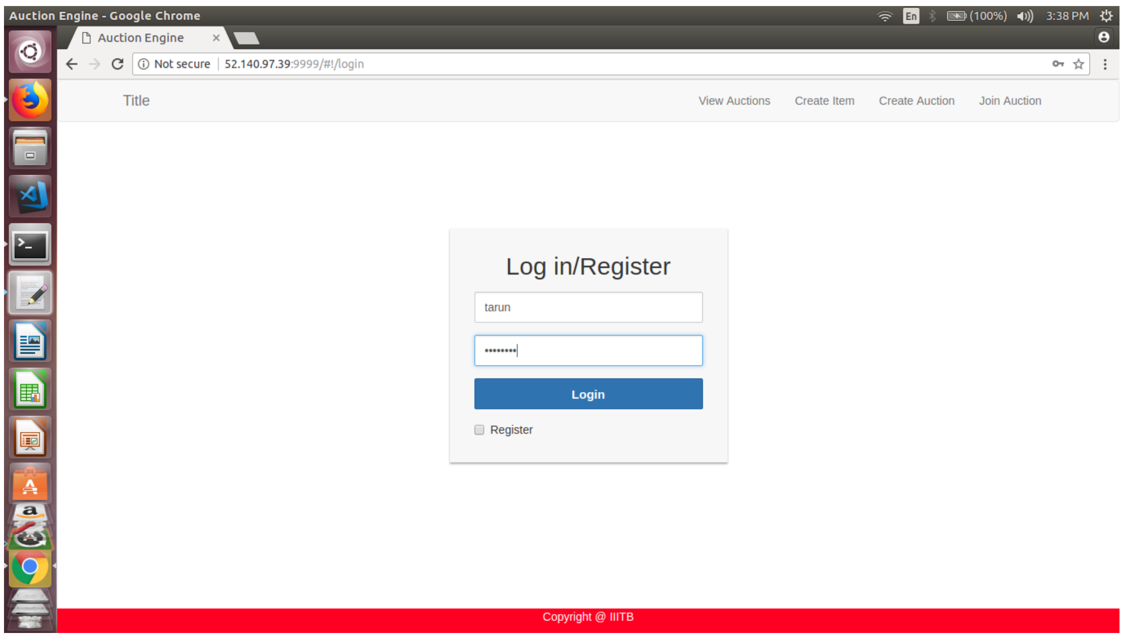
1. **Auctioneer** : In this role, user can create, start, stop various auctions and view its auction history.

2. **Bidder :** In this role, user can bid on auctions that are currently running.

<u>Basic Workflow</u> :

- **Start :** User can be either Auctioneer Or Bidder for a particular auction.
- **Auctioneer :** Auctioneer can Create Auction Items, Auctions.Only he has the permission to START, STOP the auctions created by him..
- **Bidder :** Bidder can Participate in auctions and place bid in them.
- **Item:** It is the item to be Auctioned.
- **Start Auction:** A function that starts the auction, initiated by auctioneer,if auction has ended or is currently running, error will be thrown.Users can start bidding once auction has been started.
- **Stop Auction:** A function that stops the auction(may or may not be auctioneer initiated).,if auction has not started or is finished, error will be thrown.no more bids are accepted if auction is stopped.
- **Auction Registry:** contains data for all auctions.
- **Place Bid:** Used to place bid auction.Logic of place bid varies as per auction type in consideration.if auction has not started yet or is Finished or if bid does not meet auction criteria, error is thrown.
- **Winning Bid:** decides which bid has won the auction, Logic depends upon the auction in consideration.if no winner, then winning bid is empty.this is initiated after auction is stopped
- **Item Sold:** if winning bid exists, then item is sold to the bidder of that bid else item remains unsold.
- **End:** auction Ends after item sold step.

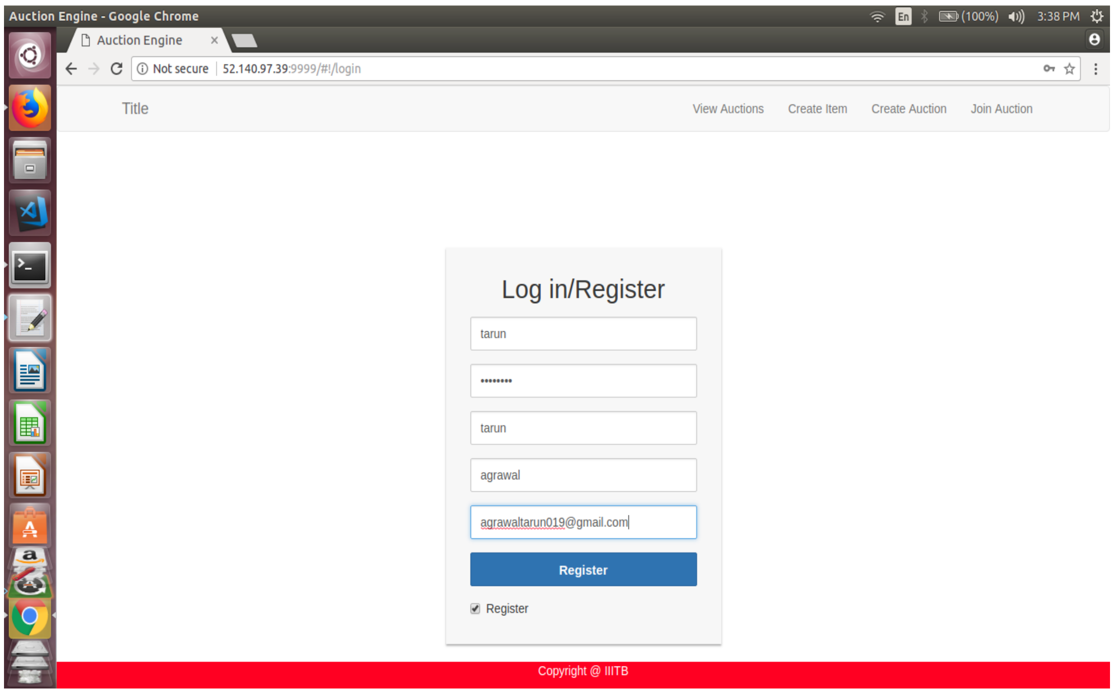
4. Screen shots



tarun

••••••••

Figure 4.0: Login



tarun

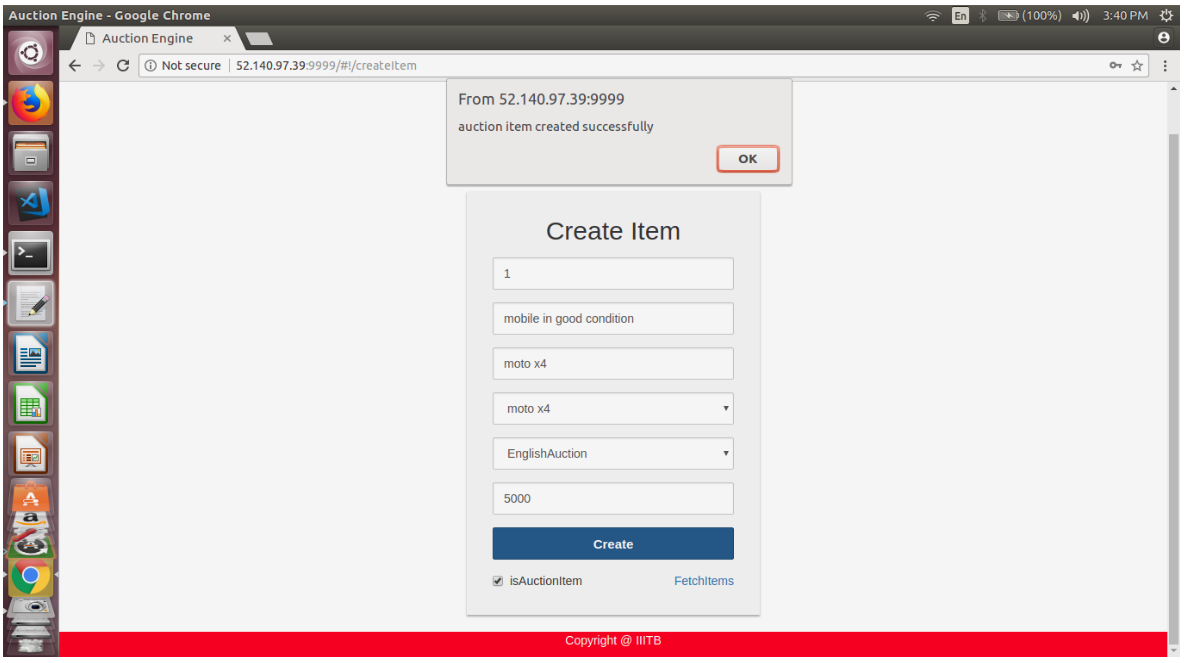
••••••••

tarun

agrawal

agrawaltarun019@gmail.com

Figure 4.1: Register User Page


Auction Engine - Google Chrome
3:40 PM
Auction Engine
Not secure
52.140.97.39:9999/#!/createItem

From 52.140.97.39:9999

auction item created successfully

OK

Create Item

1

mobile in good condition

moto x4

moto x4

EnglishAuction

5000

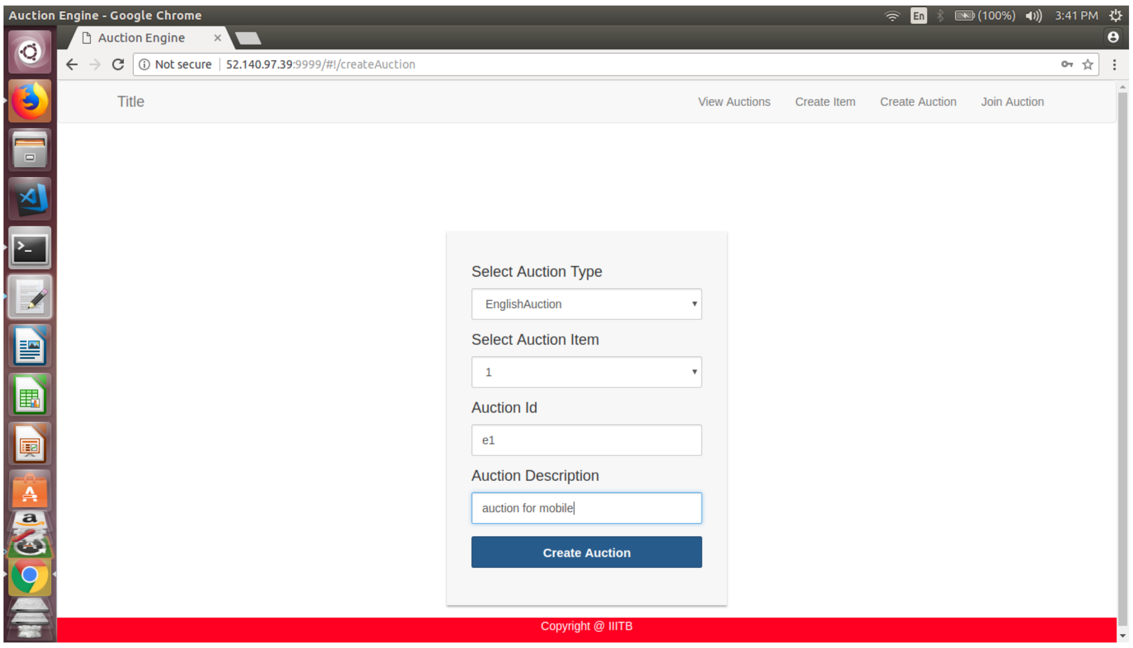
Create

☑ isAuctionItem

FetchItems

Copyright @ IIITB

Figure 4.2: Create Auction Item



Select Auction Type: EnglishAuction

Select Auction Item: 1

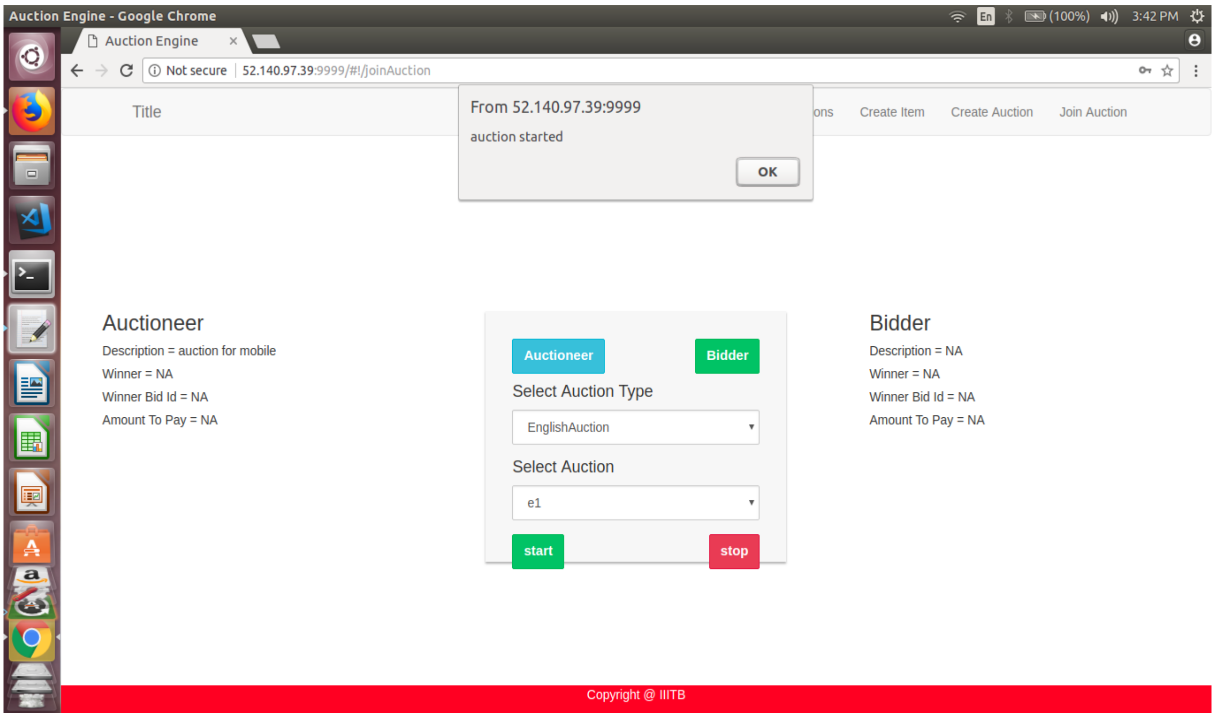
Auction Id: e1

Auction Description: auction for mobile

Create Auction

Copyright @ IIITB

Figure 4.3: Create Auction Page



From 52.140.97.39:9999

auction started

Auctioneer

Description = auction for mobile
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

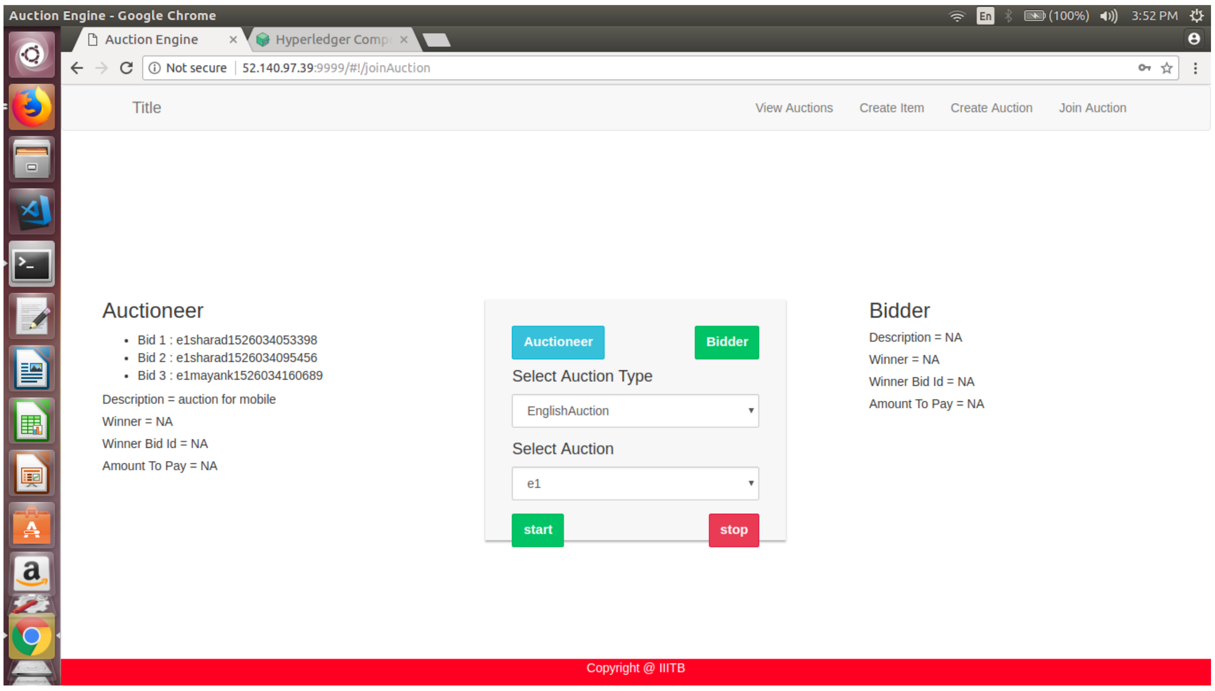
Select Auction Type: EnglishAuction

Select Auction: e1

Bidder

Description = NA
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

Figure 4.4: Join Auction page



Auctioneer

- Bid 1 : e1sharad1526034053398
- Bid 2 : e1sharad1526034095456
- Bid 3 : e1mayank1526034160689

Description = auction for mobile
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

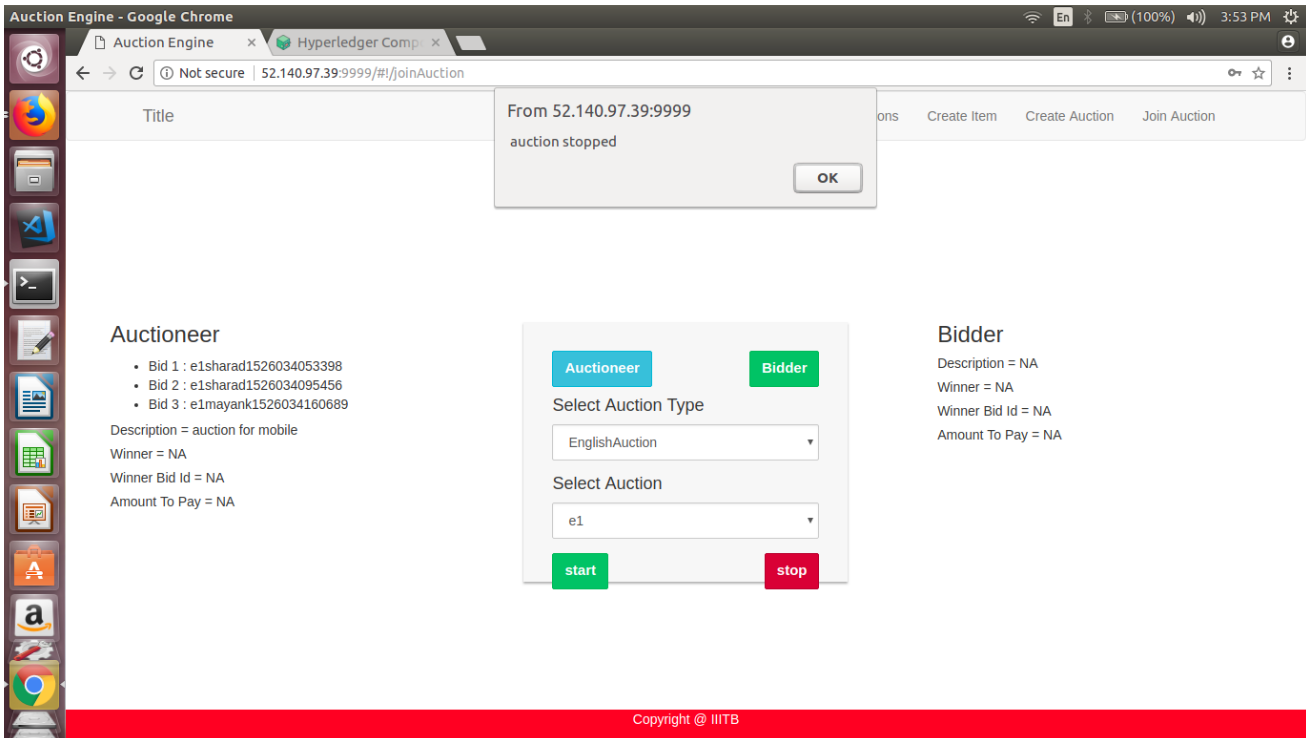
Select Auction Type: EnglishAuction

Select Auction: e1

Bidder

Description = NA
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

Figure 4.5: Join Auction(Auctioneer View)



From 52.140.97.39:9999

auction stopped

OK

Auctioneer

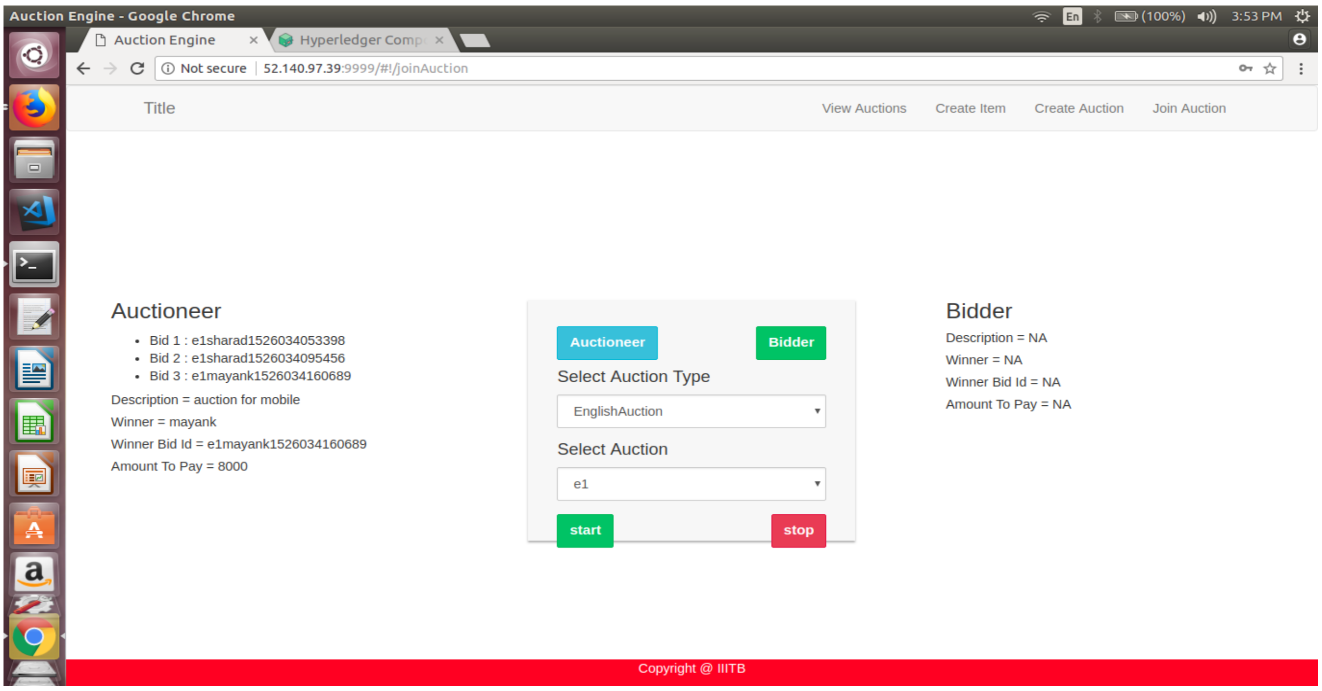
- Bid 1 : e1sharad1526034053398
- Bid 2 : e1sharad1526034095456
- Bid 3 : e1mayank1526034160689

Description = auction for mobile
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

Bidder

Description = NA
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

Figure 4.6: On Auction Stop



Auctioneer

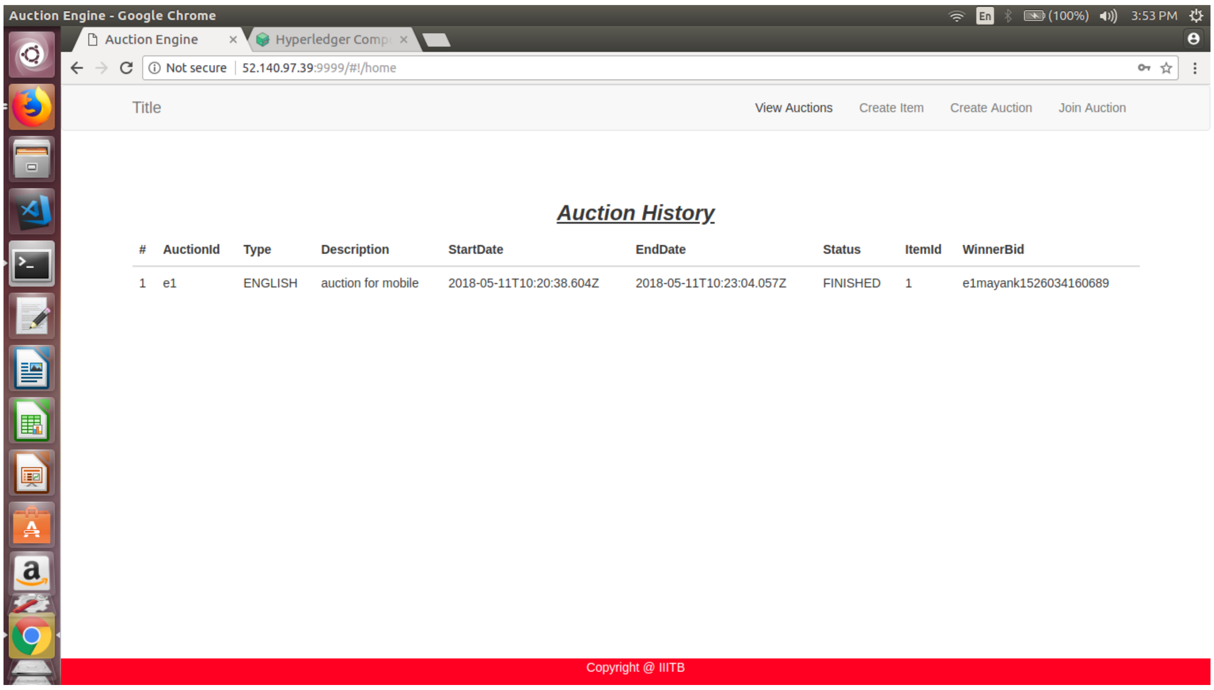
- Bid 1 : e1sharad1526034053398
- Bid 2 : e1sharad1526034095456
- Bid 3 : e1mayank1526034160689

Description = auction for mobile
Winner = mayank
Winner Bid Id = e1mayank1526034160689
Amount To Pay = 8000

Bidder

Description = NA
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

Figure 4.7: On Place Bid

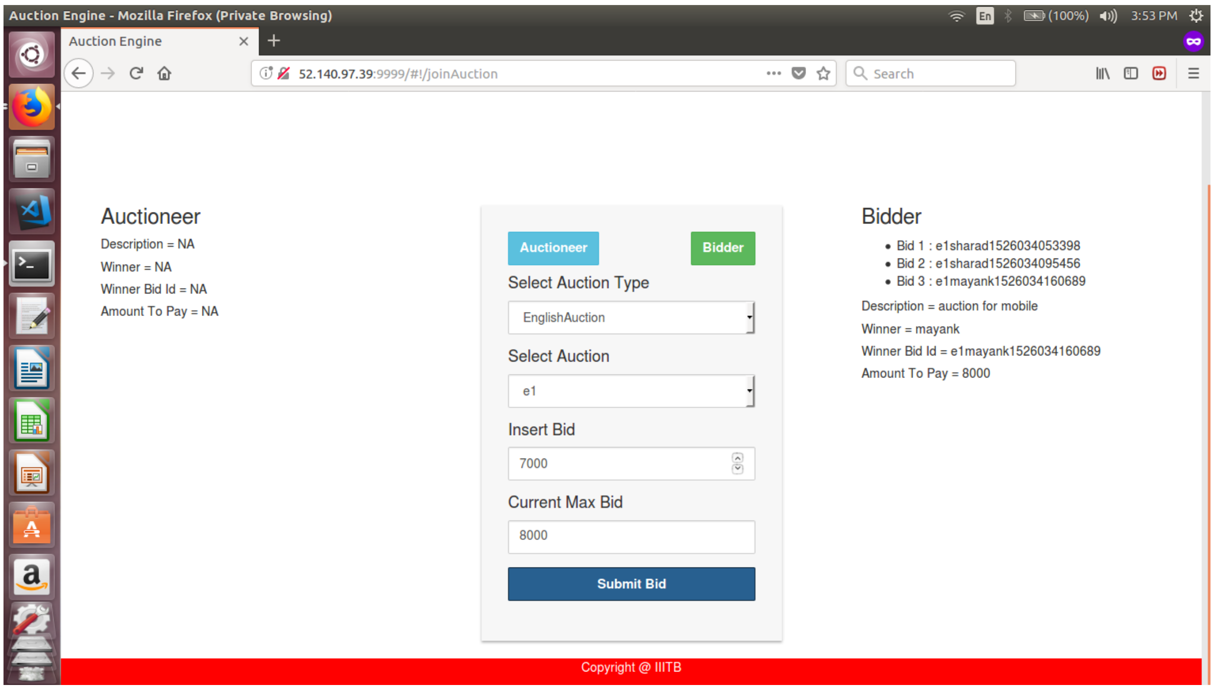


Auction History

#	AuctionId	Type	Description	StartDate	EndDate	Status	ItemId	WinnerBid
1	e1	ENGLISH	auction for mobile	2018-05-11T10:20:38.604Z	2018-05-11T10:23:04.057Z	FINISHED	1	e1mayank1526034160689

Copyright @ IIITB

Figure 4.8: Auction History details page


Auction Engine - Mozilla Firefox (Private Browsing)
Auction Engine
52.140.97.39:9999/#!/joinAuction
Search
Auctioneer
Bidder
Submit Bid

Auctioneer

Description = NA
Winner = NA
Winner Bid Id = NA
Amount To Pay = NA

Select Auction Type: EnglishAuction
Select Auction: e1
Insert Bid: 7000
Current Max Bid: 8000

Bidder

- Bid 1 : e1sharad1526034053398
- Bid 2 : e1sharad1526034095456
- Bid 3 : e1mayank1526034160689

Description = auction for mobile
Winner = mayank
Winner Bid Id = e1mayank1526034160689
Amount To Pay = 8000

Copyright @ IIITB

Figure 4.9: Join Auction(Bidder View)

5. Challenges and Future Scope

- Hyperledger fabric is still in the incubation status, hence the stability of the fabric is not proven.
- Relatively new technologies used and hence lack of online tutorials.
- we want to make it open source.
- Support more types of auctions.
- Allow user to create auction rules dynamically.

References

[1] https://github.com/angrbrd/hyperledger-fabric-basic

[2] https://www.ibm.com/blockchain/

[3] https://www.codementor.io/nodejs/tutorial

[4] https://tour.golang.org/basics/1

6. Appendices

6.1 Why Blockchain

A blockchain is simply a method of structuring data which allows a digital ledger of transactions to be created and shared amongst the participants of the network via a distributed network of computers. By using public/private key cryptography (a method of secure communication), similar to the technology that website SSL certificates use, network participants are able to add data to the online ledger without the requirement of a central authority to authenticate or manage users and data. Users can append (add) additional data to the end of the blockchain ledger in what is known as a block. It is impossible to change or remove data from the ledger once encoded into a block without getting noticed. When data gets added in this manner all the network participants (all of whom have copies of the blockchain) validate the new data via a series of automated computer processes. If the majority of network participants successfully validate the new data against the blockchain history, then the newly added data will be approved and the new block will be appended to the chain. This process forms the what is known as a consensus mechanism. Although in most use cases the term blockchain is used to refer to a publicly distributed ledger with a distributed consensus mechanism, this is not the case with all blockchain technologies. There are a range of distribution options and consensus mechanisms, these will depend on the particular use case. The example that most people know is called bit-coin. Bitcoin is public and permissionless, in that anyone can view the blockchain ledger information and anyone can interact with the network and add valid transactions to the ledger. There are also permissioned blockchains in existence, in these systems the network is comprised only of known or permissioned participants. This may be desirable in some use cases where users must be authenticated and/or verified. A permissioned or private blockchain is one that can only be accessed by pre-approved parties. Permissioned blockchains facilitate many of the the advantages of digital assets powered by public blockchains fast and cheap transactions permanently recorded in a shared ledger without the troublesome openness of the public network where anyone can be a node on the network anonymously. For our use case we will be requiring a permissioned or private blockchain network where each participant (the guard) will be first approved to log into the system so that he could also become a part of the blockchain network.

6.2 How Blockchain protects the data

We store the head hash pointer of the blockchain.

Whenever an adversary tries to tamper with data of a block, the hash stored in the next block doesnt match now with the hash of the tampered block.

The adversary can try to tamper with the next block as well, but this will lead him to tamper with the next block as well, and the process goes on until he has to change the head hash pointer.

The head hash pointer is always stored with us hence we will immediately notice any tampering done with the head hash pointer.

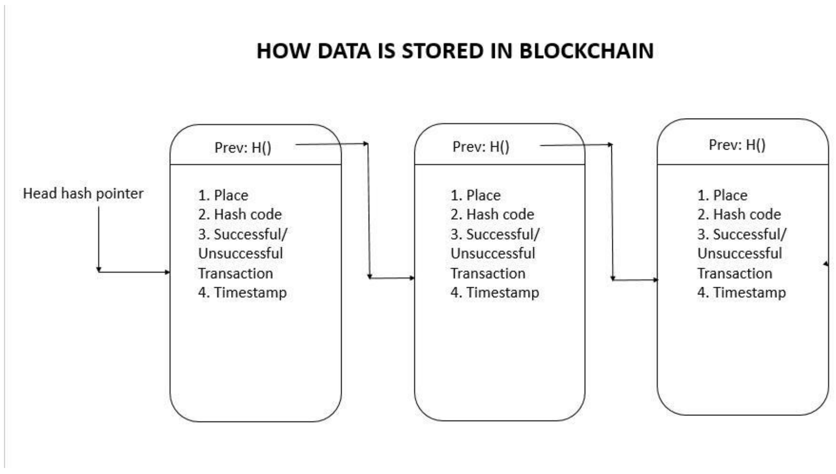

HOW DATA IS STORED IN BLOCKCHAIN
Head hash pointer
Prev: H()
1. Place
2. Hash code
3. Successful/ Unsuccessful Transaction
4. Timestamp
Prev: H()
1. Place
2. Hash code
3. Successful/ Unsuccessful Transaction
4. Timestamp
Prev: H()
1. Place
2. Hash code
3. Successful/ Unsuccessful Transaction
4. Timestamp

Figure 6.1: How data is stored in Blockchain

6.3 Hyperledger Fabric

Hyperledger Fabric is a platform for distributed ledger solutions, underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility and scalability. It is designed to support pluggable implementations of different components, and accommodate the complexity and intricacies that exist across the economic ecosystem. Business networks where we do not wish to share the inner de-tails of the business deals require a private blockchain network unlike bitcoin where everything is transparent and public. Hyperledger fabric enables us to keep the inner details of a deal concealed from the rest of the world. Hyperledger Fabric deliv-ers a uniquely elastic and extensible architecture, distinguishing it from alternative blockchain solutions.

6.4 Hyperledger Functionalities

Some of the salient features in Hyperledger Fabric are :

Peers - The peers participate in common consensus among themselves in accordance to the chaincode deployed in the network and decide whether a given transaction has to be include in a new block or not. In our case the peers were the administrators of the campus and the supervisors of the guards.

Ledger - The immutable, shared ledger encodes the entire transaction history for each channel, and includes SQL-like query capability for efficient auditing and dispute resolution. We can query the hash of a particular block with the command -

```
curl < docker_host_ip >: 7050=chain=blocks=Block_number
```

Privacy through Channels - Channels enable multi-lateral transactions with the high degrees of privacy and confidentiality required by competing businesses and regulated industries that exchange assets on a common network.

Security & Membership Services - Permissioned membership provides a trusted blockchain network, where participants know that all transactions can be de-tected and traced by authorized regulators and auditors.

Chaincode –Chaincode is the core part of the Hyperledger Fabric. With chain-code only we can decide what should go into our blocks of the blockchain. Basically chaincode is the contract signed by the parties who want to have a business deal with each other but do not trust each other hence need the func-tionalities of the blockchain.

There are three aspects to chaincode development:

Chaincode

Interfaces APIs

Chaincode Responses

6.5 Chaincode Interfaces

A chaincode implements the Chaincode Interface that supports two methods[2]: There are three aspects to chaincode development:

Init()

Invoke()

Query()

6.5.1 Init()

Init is called when you first deploy your chaincode. As the name implies, this function is used to do any initialization your chaincode needs. We initialize the guards name as the key and append null values.

6.5.2 Invoke()

Invoke is called when you want to call chaincode functions to do real work (write to the ledger). Invocations are captured as transactions, which get grouped into blocks on the chain. When you need to update or query the ledger, you do so by invoking your chaincode. In our use case we first query the current value of the guards scans and append the new record at the end of those retrieved records and put it back to the blockchain.

6.5.3 Query()

When we want to need to know the value of a particular asset, which we are tracking in our blockchain. In our use case we query the successful scans and un-successful scans of each guard.

6.6 Dependencies to be taken care of in chaincode

The import statement lists a few dependencies for the chaincode to compile successfully.

fmt –contains Println for debugging/logging.

erors –standard go error format.

shim –contains the definitions for the chaincode interface and the chaincode stub, which are required to interact with the ledger.

6.7 Chaincode API

When the Init or Invoke function of a chaincode is called, the fabric passes the stub shim.ChaincodeStubInterface parameter and the chaincode returns a pb.Response. This stub can be used to call APIs to access to the ledger services, transaction con-text, or to invoke other chaincodes.

The current APIs are defined in the shim package, and can be generated with the following command:

godoc github.com/hyperledger/fabric/core/chaincode/shim

However, it also includes functions from chaincode.pb.go (protobuffer functions) that are not intended as public APIs.

6.8 Response

The chaincode response comes in the form of a protobuffer.

message Response f

// A status code that should follow the HTTP status codes. int32 status = 1;

// A message associated with the response code. string message = 2;

// A payload that can be used to include metadata with this response. bytes payload = 3;

g You can interact with chaincode either through Command-Line Interface or the SDK(Software Development Kit)