# Summary

Blockchain is a promising technology to solving some of the challenges faced by traditional e-commerce. Security of confidential consumer and company data, political and geographical restrictions, traceability and accountability to shareholders, and higher levies on transactions are problems faced in traditional e-commerce. This provides an opportunity for a platform based on blockchain to be used to provide accountability and traceability to shareholders due to immutability of data, lower the charges levied on transaction, while facilitating payment via crypto-currency that tackles the challenges of political and geographical retractions.

Development of the project was based on Ethereum, using Solidity to code the smart contract at the heart of this project. Mappings were found to be important data structures in the development of state variables to hold business and purchases information. Solidity’s modifiers came in handy in providing access control to parties making use of the platform. Ganache was used to create a local Ethereum network on which the smart contract was tested and deployed using Truffle. A client was created to interact with the smart contract from the browse by leveraging on Web3.js, and creating a server via Node.js. HTML, JavaScript and CSS were extensively used in the development of the client.

The platform so developed was able to allow a merchant to register their business on the platform. A user could view a list of the businesses registered on the platform. The specific businesses owned under shareholding of a particular user could be viewed independently of the entire inventory of businesses registered on the platform. A merchant was able to add a shareholder to their business, allowing them to view the transaction histories and balances related to that business. The merchant could withdraw funds held in escrow by the platform into their wallets.

Evaluation of the project revealed that the primary objectives set out by the project had been met. A few areas, however, were found where future work could be done on the project, including: improvement on the UX/UI, analytics functions and penetration testing of the smart contract.

Table of Contents

[Summary i](#_Toc141396700)

[Table of Figures iv](#_Toc141396701)

[Table of Tables v](#_Toc141396702)

[Proposed System 1](#_Toc141396703)

[Overview 1](#_Toc141396704)

[Functional Requirements 1](#_Toc141396705)

[Non-functional requirements 1](#_Toc141396706)

[Context Diagram 2](#_Toc141396707)

[System External 2](#_Toc141396708)

[System Models 3](#_Toc141396709)

[Software Used 5](#_Toc141396710)

[Implementation 6](#_Toc141396711)

[Contracts folder 7](#_Toc141396712)

[Data Structures Used 7](#_Toc141396713)

[Smart Contract Modifiers 10](#_Toc141396714)

[Smart Contract Functions 10](#_Toc141396715)

[build\contracts folder 12](#_Toc141396716)

[client folder 12](#_Toc141396717)

[test folder 14](#_Toc141396718)

[Results 15](#_Toc141396719)

[Evaluation of the project 24](#_Toc141396720)

[Gateway for payments to businesses 24](#_Toc141396721)

[Storing of transactions 24](#_Toc141396722)

[Display of transactions 25](#_Toc141396723)

[Usability 25](#_Toc141396724)

[Security 26](#_Toc141396725)

[Recommendations 27](#_Toc141396726)

[Appendix 28](#_Toc141396727)

# Table of Figures

[Figure 1: Context diagram 11](#_Toc97505396)

[Figure 2: Project directory tree 17](#_Toc97505397)

[Figure 3: Contracts folder 18](#_Toc97505398)

[Figure 4: Data structure used to store business information 19](#_Toc97505399)

[Figure 5: Data structure used to store purchases information 19](#_Toc97505400)

[Figure 6: Data structure used to store business owned by a merchant 20](#_Toc97505401)

[Figure 7: Data structure used to store businesses for a shareholder 21](#_Toc97505402)

[Figure 8: Accessing the smart contract from a client 24](#_Toc97505403)

[Figure 9: Ganache accounts instance 26](#_Toc97505404)

[Figure 10:Deploying the smart contracts on the local network 27](#_Toc97505405)

[Figure 11:Starting Node.js 28](#_Toc97505406)

[Figure 12: Registering a new business to the platform 28](#_Toc97505407)

[Figure 13:Viewing all businesses registered on the platform 29](#_Toc97505408)

[Figure 14:Simulating payments to Bingo Bangers 30](#_Toc97505409)

[Figure 15:Merchant viewing his businesses 30](#_Toc97505410)

[Figure 16:Viewing transaction history of a business 31](#_Toc97505411)

[Figure 17: Adding shareholder to business 32](#_Toc97505412)

[Figure 18:Page reflecting addition of shareholder to business 32](#_Toc97505413)

[Figure 19:Shareholder viewing his/her businesses 33](#_Toc97505414)

[Figure 20:Shareholder viewing transaction history and balance of business 33](#_Toc97505415)

[Figure 21:Business and wallet balance before withdrawing ether from business 34](#_Toc97505416)

[Figure 22:Business and wallet balance after withdrawal from the business 34](#_Toc97505417)

[Figure 23: Project Gantt Chart 40](#_Toc97505418)

# Table of Tables

[Table 1: Registering Business 12](#_Toc97505419)

[Table 2: Making Payment 12](#_Toc97505420)

[Table 3: Viewing payment history 13](#_Toc97505421)

[Table 4: Clients folder 24](#_Toc97505422)

## Proposed System

### Overview

The proposed system seeks to make use blockchain in facilitating easy integration of crypto-currency payment in e-commerce websites. This shall reduce the fee charged on users per transaction while providing a more robust security mechanism compared to traditional payment systems. The system shall maintain a record of payments made to merchant accounts on the blockchain. This shall provide immutability of the transaction data stored, providing integrity. The system shall allow authorized users to track the history of payments made to their relevant accounts. This shall provide availability, while fostering traceability of the financial records.

### Functional Requirements

1. The system shall allow payment of e-commerce transactions by crypto-currency. This shall reduce the charge levied on transactions on the e-commerce platforms from 2-3% of every transaction to $0.01 of every transaction.
2. The system shall immutably store the transactions made via the system on the blockchain.
3. The system shall allow shareholders to chronologically trace the transactions made via the system. This shall improve accountability of the e-commerce business by displaying a record and consequently sum of the revenue streamed into the business via the system.

### Non-functional requirements

**Usability**

* The number of web pages accessed to fulfill any functional requirement should not exceed 4.
* A user should be able to initialize crypto-currency payment via the system within 10 seconds after the original setup.

**Security**

* The system is to only display a trace history of the payments made to a business account to authorized users.

**Interfaces**

The system must:

* Interface Ethereum to browsers using web3.js
* Interface business logic with the web client using scripting language

### Context Diagram

The context diagram for this project is shown in Figure 1 below.

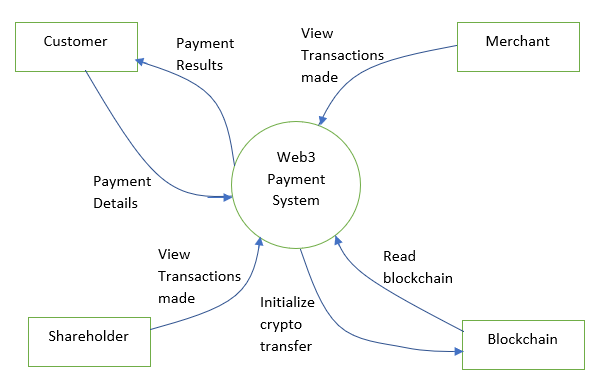


Figure 1: Context diagram

### System External

Customer

A customer is a buyer on an e-commerce platform making use of this payment system. The customer is capable of making payments using crypto-currency during checking out. The value of the items in his/her shopping cart/ invoice are capable of being paid for by this system.

A merchant

A merchant is the party offering products or services on the e-commerce platform. The business owner is capable of receiving payments to his/her account by crypto-currency using the system. The merchant is also capable of viewing the history of payment made to his/her account on the system.

Shareholder

A shareholder is any party invested in the business making payments on the e-commerce platform. The shareholder is privileged to view the books of the business, as such the history of payments made to his/her invested business.

### System Models

**Use Case Model**

Registering Business

|  |  |
| --- | --- |
| Summary | This use case allows a merchant to register his/her e-commerce business on the service. |
| Basic Flow: | 1. The use case starts when the merchant requests to register his/her business on the service.  2. The merchant is requested information regarding his/her e-commerce website such as business name, description and call-back URLS.  3. The business details are added onto the blockchain.  4. The merchant is provided feedback on the status of his/her registration. |
| Preconditions: | The merchant has a crypto account. |

Table 1: Registering Business

Making Payment

|  |  |
| --- | --- |
| Summary: | This use case allows a user to make e-commerce purchase using the system |
| Basic Flow: | 1. The use case starts when the user clicks the checkout button on the e-commerce website  2. The user is prompted to enter account details  3. Funds are transferred from the user’s account to the merchant account |
| Preconditions: | The user has a crypto-currency wallet |

Table 2: Making Payment

Viewing History of Payments

|  |  |
| --- | --- |
| Summary | This use case allows authorized stakeholders to view the history of payments made to their related business accounts through the system |
| Basic Flow: | 1. The use case starts when the shareholder requests to view the history of payments made to their accounts.  2. The user privilege to access the information shall be checked  3. Financial records related to the shareholder’s account are read from blockchain  4. The records shall be displayed back to the shareholder |
| Alternative Flows: | 3. The user shall be notified of lack of privilege in case of denied access |
| Preconditions: | The merchant account exists. |

Table 3: Viewing payment history

# Software Used

* Ethereum
* Ganache
* Truffle
* Web3.js
* Solidity
* Node.JS

## 

# Implementation

Different tools and frameworks were used in the development of this project. The document root for the project is shown:

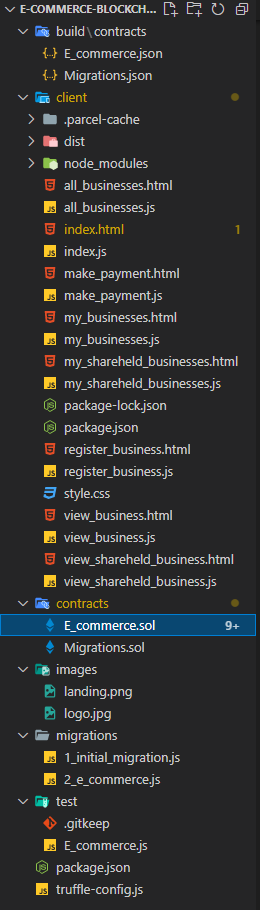


Figure : Project directory tree

A description of the significant files in the project directory is as follows

## Contracts folder

This folder contained the smart contract deployed on the local test network. The smart contracts were coded in solidity. A brief description of the files in this directory is as follows:

|  |  |
| --- | --- |
| File | Purpose |
| E\_commerce.sol | This was the actual smart contract coded for the project. The data structures needed to store and facilitate information retrieval, and business logic functions to achieve the project’s objectives were coded here |
| Migrations.sol | This was a default smart contract used in the truffle suite to help in deployment of other smart contracts |

Figure : Contracts folder

The E\_commerce.sol is the heart of this project. This is the file that contains the smart contract that makes this project possible. As such, a description of the fundamental data structures and functions that make up the smart contract is highly important. The following sections seeks to explain the significance of these elements to the project.

### Data Structures Used

The persistent variables used to store data on smart contracts are known as state variables. Different state variables were used in this project to store different forms of information. The significant state variables used for this project where:

**1. Storage of business information**

A mapping was used to store business information. The key of the mapping was an integer that was the businessId and the value a structure that had different fields.

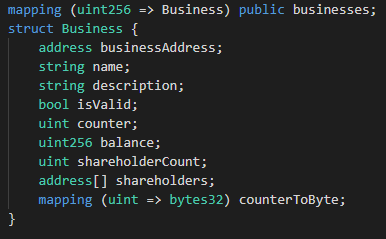


Figure : Data structure used to store business information

The structure stored: the Ethereum address of the merchant owning the business, name and description of the business, a validity flag for internal book-keeping of the smart contract, a counter for the number of transactions made to the business, the balance of the business, a counter for the number of shareholders of the business, an array for the addresses of the shareholders of the business and a mapping for transposing the internal transactionId used in the smart contract to the transactionId of any data type provided by any third party making payment to the business.

**2. Storage of purchases made to a business**

A nested mapping was used to store the purchases made to businesses registered on the service. The value of the outer nest was the businessId with the value being another mapping. The value of the inner mapping was the purchaseId of the transaction made, and the value a structure containing information about the purchase.

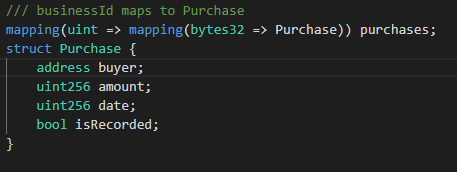


Figure : Data structure used to store purchases information

The structure contained the Ethereum address of the party that made payment to the business, the amount transferred, the date the payment was made, a recorded flag for internal book-keeping of the smart contract.

In addition to these two critical state variables, two other convenience state variables were defined to provide crucial redacted information about the two state variables above. The use of these two extra state variables was deemed necessary to eliminate the need for extensive iteration on the previous state variables. This is because unlike other programming language, gas is consumed for every iteration in Solidity. As the size of the previous state variables grew and select information was needed from the state variables that could otherwise only be obtained through iterating over the state variables, an incredible amount of gas would have to be consumed. If this gas exceeded the gas limit defined for that block, the transactions would have to be dropped.

3. Storing the businesses owned by a particular merchant

A mapping was used to store the businesses owned by a particular merchant. The key of the mapping was the Ethereum address of the merchant, and the value a structure containing information about the businesses he/she owned.

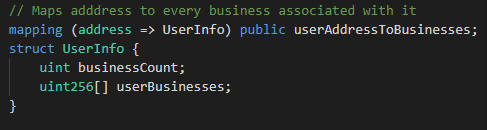


Figure : Data structure used to store business owned by a merchant

The structure contained an integer storing the number of businesses the merchant owned, and an array containing the businessIds of his/her businesses.

4. Storing the businesses owned by a particular shareholder

A mapping was used to store the businesses owned by a particular merchant. The structure of the state variable was similar to the former.



Figure : Data structure used to store businesses for a shareholder

### Smart Contract Modifiers

Modifiers in Solidity are function called prior to execution of a function to perform repetitive actions such as data checks. Different modifiers were used in this smart contract as described.

modifier isOwner(uint256 \_businessId) {

This modifier checked that the person invoking a function on the smart contract was the actual owner of the business he/she wanted to get/ post information about. The function took the businessId as argument. This function ensured that only authorized parties had access to functions dealing with confidential business information.

modifier isOwnerOrShareholder(uint256 \_businessId){

The modifier above checked that the person invoking a function for a particular business was the merchant owner or shareholder of the business

modifier businessExists(uint256 \_businessId){

The modifier above checked that the businessId was for a valid business defined on the smart contract.

modifier userExists(){

The modifier checked that the account making a call to the smart contract belonged to a merchant stored in the smart contract.

### Smart Contract Functions

Numerous functions where coded to implement the business logic of the project in the smart contract. This section provides a brief description of the most important functions:

function registerBusiness(string memory \_name, string memory \_description) public returns (uint256){

The function above registers a new business on the smart contract. The function takes the name and description of the business, and returns the businessId of the business.

function mapUserAddressToCreatedBusiness(uint256 \_businessId, address \_sender) public {

The function above adds a business to the state variable defined in (3) of the previous section used to store the businesses owned by a particular merchant. The function takes the businessId and Ethereum address of the merchant.

function getAllBusinesses() public view returns (string [3][] memory){

The function above fetches an array of the business registered on the blockchain.

function mapShareHolderToMyBusiness(uint256 \_businessId, address \_shareholder) public businessExists(\_businessId) isOwner(\_businessId){

The function above was invoked to add a shareholder to a merchant’s business. The function takes the businessId and Ethereum address of the shareholder as parameters.

function getUserBusinesses() public view returns (string [4][] memory){

The function above fetches the businesses owned by a particular merchant

function getBusinessShareholders(uint256 \_businessId) public view businessExists(\_businessId) returns (string[] memory){

The function above fetches an array of the shareholders registered by a merchant to a particular business.

function makePayment(uint256 \_businessId, bytes32 \_purchaseId, uint256 \_date) external businessExists(\_businessId) payable{

The function above makes a payment to a particular business. The function takes the businessId, purchaseId and date of transaction as parameters.

function sendViaCall(uint256 \_businessId, uint256 \_amount) public businessExists(\_businessId) isOwner(\_businessId) returns (bool){

The function above transfers Ethereum from the smart contract’s account that holds the businesess money in escrow to the Ethereum account of the business’s merchant.

function checkPayments(uint256 \_businessId) public view businessExists(\_businessId) isOwnerOrShareholder(\_businessId) returns (uint256[2][] memory) {

The function above fetches a 2D array having the amount and date of transactions made to a particular business.

## build\contracts folder

This folder stored the (Application Binary Interface) ABI definitions of the smart contracts deployed on the local network. The ABI is used by external applications to have knowledge of the smart contract’s interface, as such facilitating access of the smart contracts by external applications. The truffle suite automatically generated these files on migration of each corresponding smart contract.

## client folder

This folder was used to store the decentralized application’s (Dapp’s) code. Node.js was the server environment used in this project. A brief description of the files in this directory is as shown. For most of the html files defined where corresponding javascript files.

|  |  |
| --- | --- |
| File | Purpose |
| node\_modules | This folder contained modules used by Node.js |
| package.json | This file was used to store important metadata relevant to Node.js. The entry point, run scripts and dependencies used in Node.js for the project where defined in this file |
| index.html | Was the landing page and entry point of npm into the project |
| register\_business.html | This page was used by a merchant to register his/her business to the service |
| make\_payment.html | This page was used to simulate the calls made to make payment to a particular business registered on the platform. |
| all\_busineeses.html | This page was used to show all businesses registered on the service |
| my\_businesses.html | This page was used to display the businesses owner by a particular user out of all the business registered in the platform |
| view\_business.html | This page was used to show the transcations made to a particular business of a merchant. The page allowed the merchant to view his/her account balance, withdraw Ethereum into his/her wallet, and add or view the shareholders defined for his/her business. |
| my\_shareheld\_businesses.html | This page was used by a shareholder to view the business her/she had a share on |
| view\_shareheld\_business.html | This page was used by a shareholder to view the transaction history of a business he/she had a share on, and the balance to that account |

Table : Clients folder

The client folder was a blend of HTML, CSS and JavaScript. Excluding the presentation layer defined by HTML and CSS, an important aspect of the client was connecting to the smart contract deployed on the local network and having access to the Metamask wallet installed on the user’s browser. This section discusses some of the important concepts employed to this regard.

**Accessing the smart contract from the client**

In order to access the smart contract from the browser, the Web3.js framework was used.

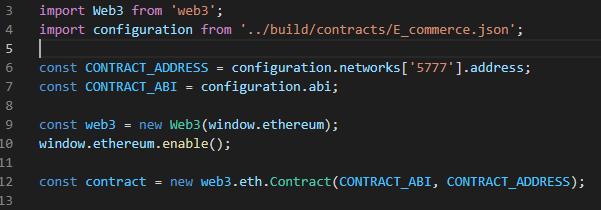


Figure : Accessing the smart contract from a client

The Web3 framework was first imported, followed by the ABI describing the smart contract. The port on which Ganache ran on the PC was specified in line6, and a Web3 object created and Ethereum link to the browser enabled on lines 9 and 10 respectively. Finally, an object was availed in line 12 that could be used to access the smart contract functions.

## test folder

This folder was used to store JavaScript tests written to test the smart contracts using Truffle suite.

# Results

A workspace was created in Ganache for the project. Ganache ran on port 5777 on the localhost. The software provided 10 accounts, each initially initialized to 100 ether. A screenshot of the Ganache instance at the moment of final testing was as shown.

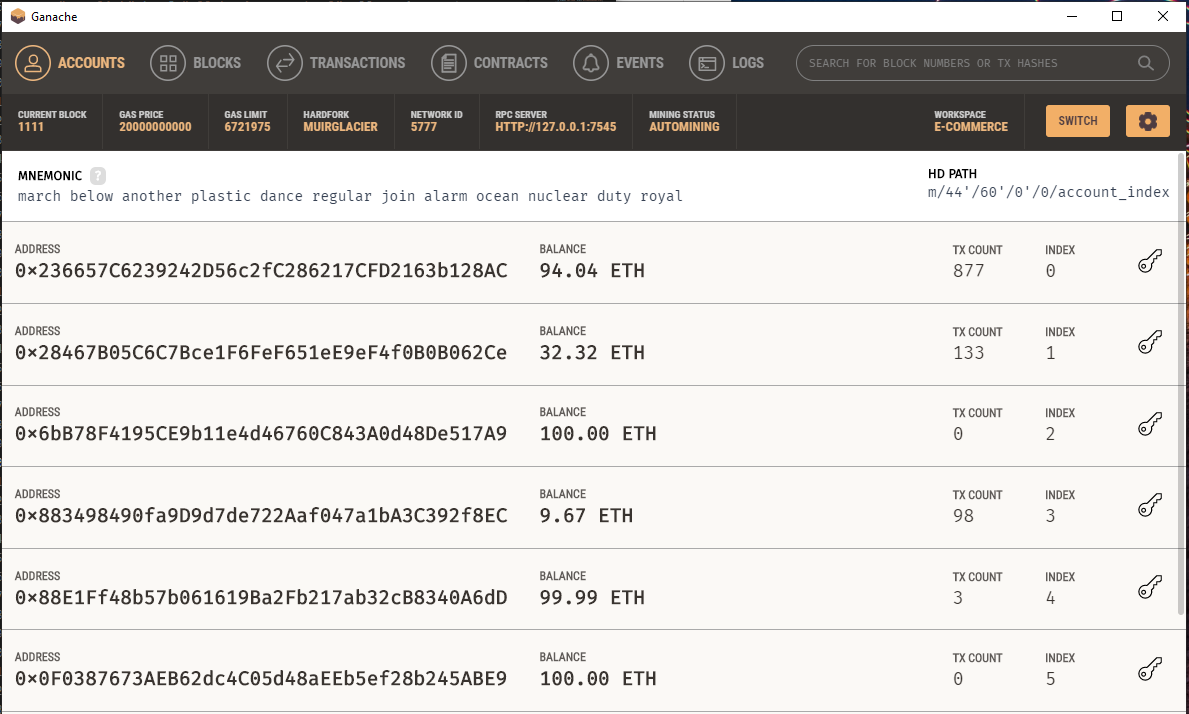


Figure : Ganache accounts instance

The smart contracts defined in the Implementation chapter were deployed on the local network supported by Ganache using the Truffle suite. The contracts were first compiled prior to deployment. The first account in Ganache above: 0x236657C6239242D56c2fc286217CFD2163b128AC was used to deploy the smart contracts. Some gas was consumed in deploying the contracts, totaling to 0.07036234 ether.

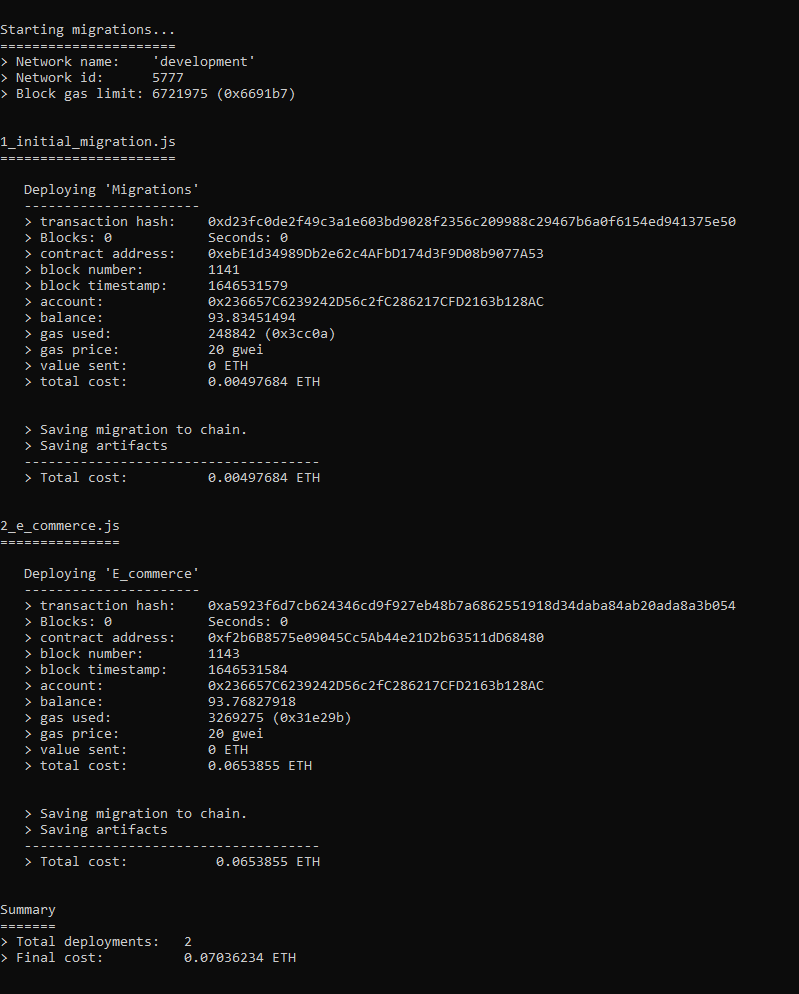


Figure :Deploying the smart contracts on the local network

The Node.js server was then started, running on port 1234 of localhost. Parcel was used to bundle the client and enable hot-swapping of content on the browser on changes.

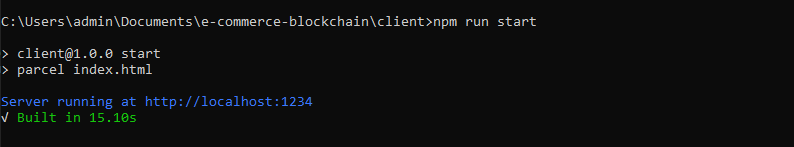


Figure :Starting Node.js

A new business was registered on the platform by the user having Ethereum address: 0x88E1Ff48b57b061619Ba2Fb217ab32cB8340A6dD. Some gas was consumed from the merchant’s account in order to create the account on the platform.

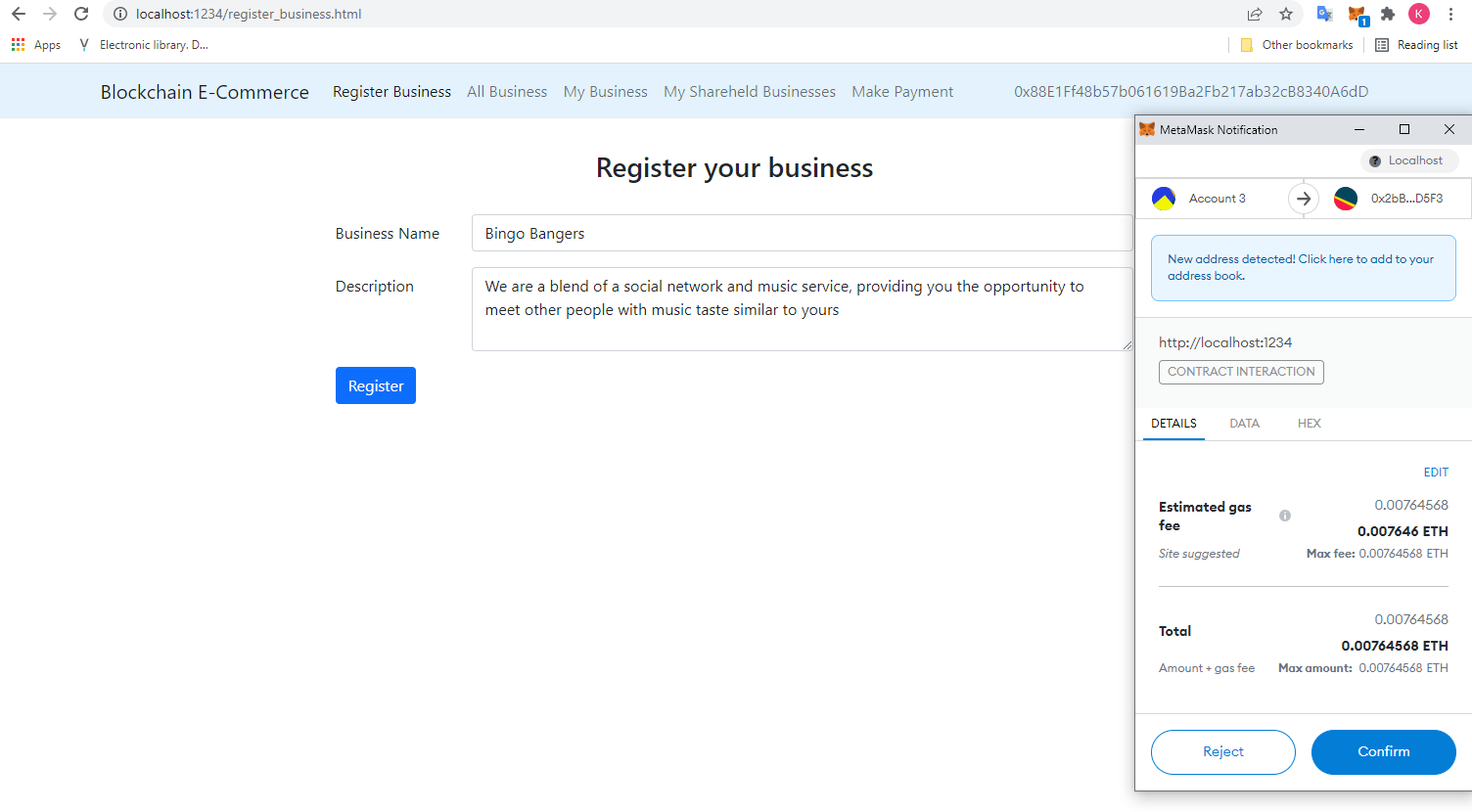


Figure : Registering a new business to the platform

Similarly, another business – Xavier Inc was created by another merchant of Ethereum address: 0xCC56C0d59C6899324D5d1Df6a2732fd4d2B9a33C.

The two businesses registered on the platform were visible from the “all\_businesses.html” page. Generic icons were used for the businesses. The business names, descriptions and Ethereum addresses of the owners were displayed from that screen.

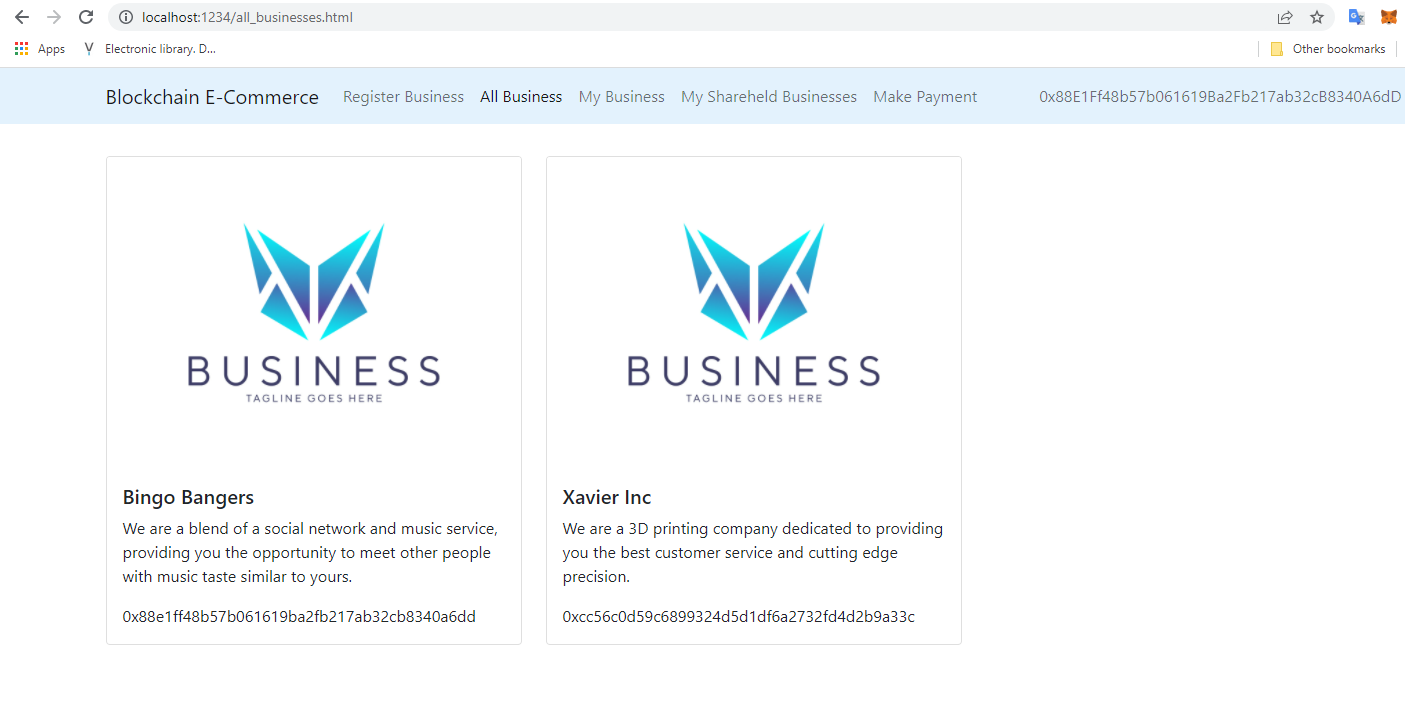


Figure :Viewing all businesses registered on the platform

Using the “make\_payments.html” page, several payments were simulated going to the “Bingo Bangers” business (business ID 1). The simulation of payments generated random dates within 5 days of the current day of payment simulation in order to facilitate the chronological display of transactions made to the business later.

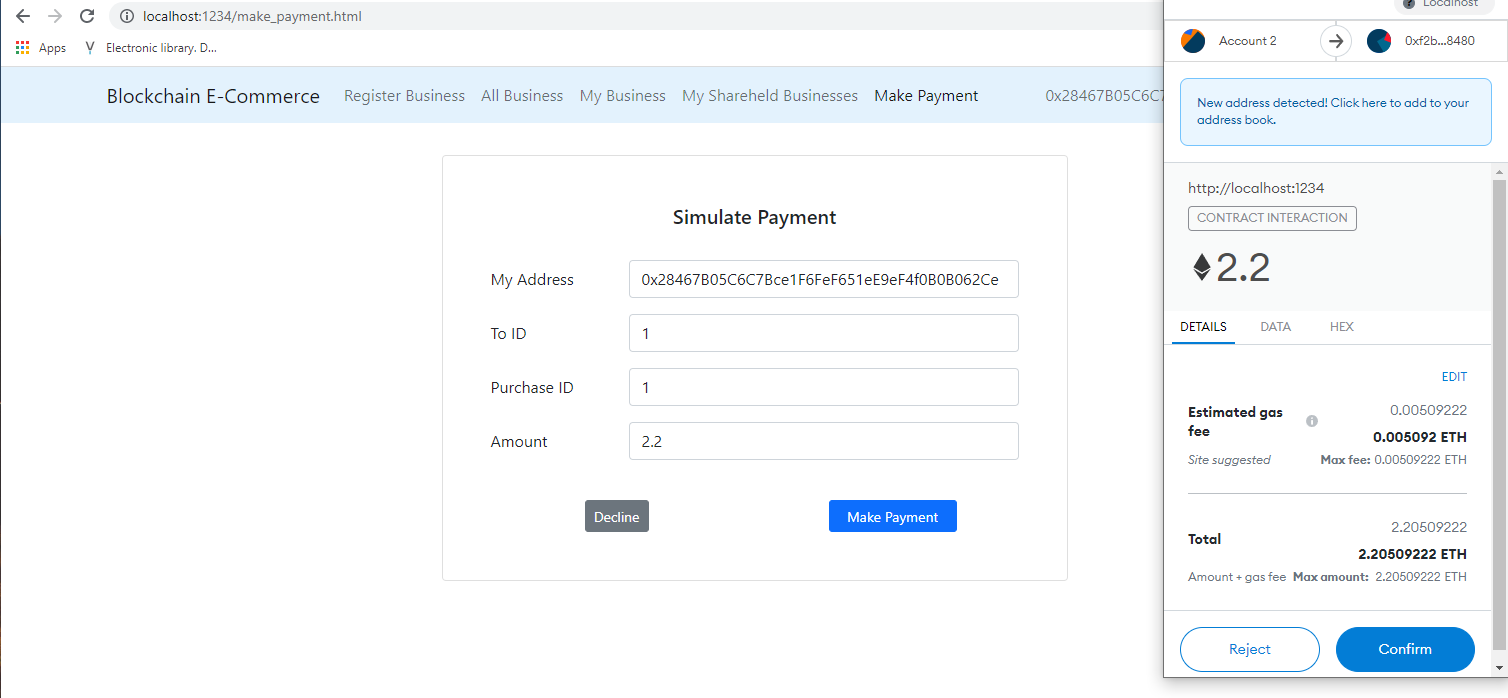


Figure :Simulating payments to Bingo Bangers

The merchant with account 0x88E1Ff48b57b061619Ba2Fb217ab32cB8340A6dD was able to view only the businesses under his account using the my\_businesses.html page. It was noted that the other business “Xavier Inc”, registered under a different account was not visible in this page since it didn’t belong to that merchant.

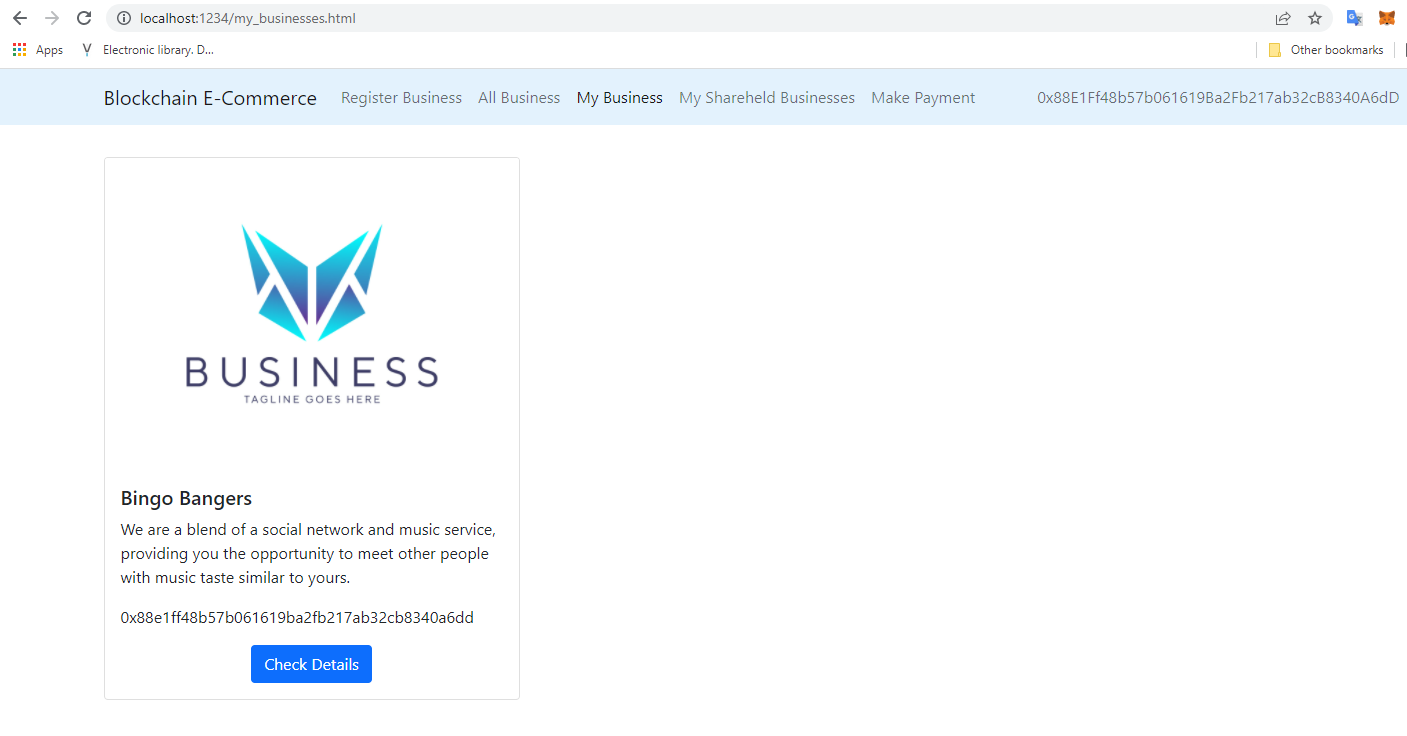


Figure :Merchant viewing his businesses

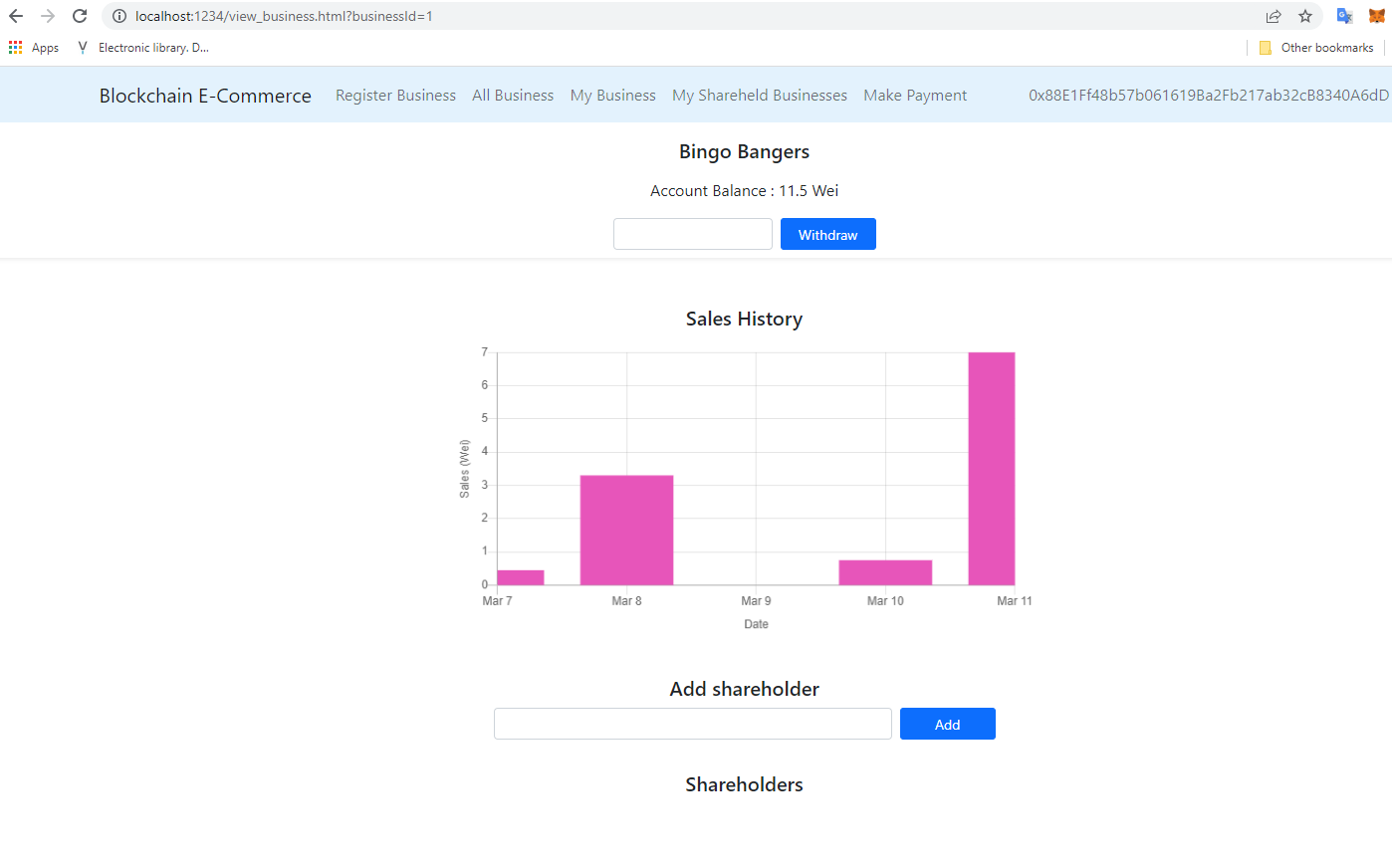
Upon clicking the “Check Details” button, the merchant was directed to a page containing detailed information about the business. The account balance for the business had increased to 11.5 Wei, and the daily transactions made to the account chronologically displayed using a bar chart. For the payments simulated, the daily sales made from March 7th to March 11th were displayed on the bar chart . 

Figure :Viewing transaction history of a business

The merchant was able to add a shareholder: 0xCC56C0d59C6899324D5d1Df6a2732fd4d2B9a33C to the business. Gas was consumed in this process.

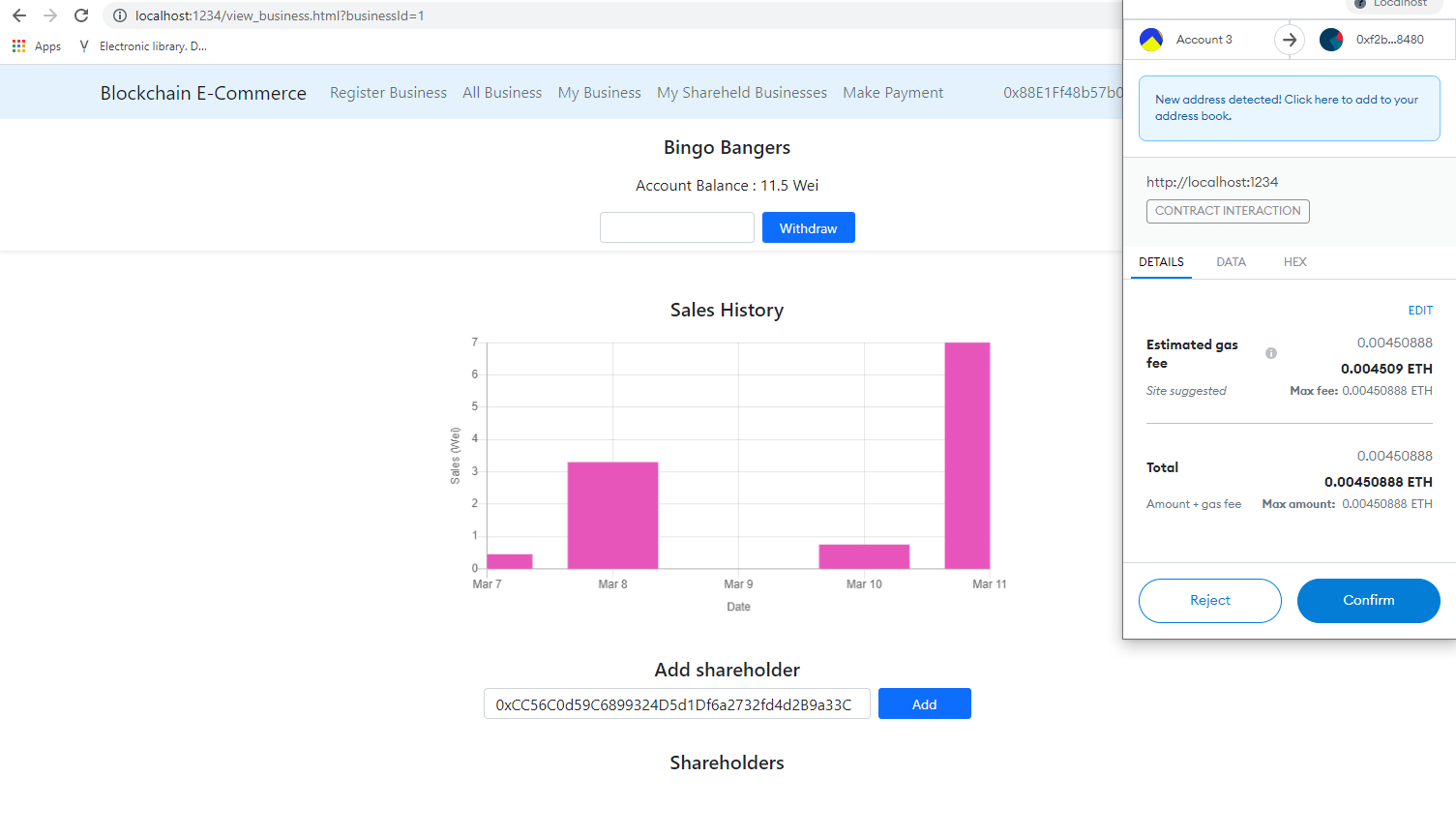


Figure : Adding shareholder to business

After confirming the process, the page refreshed, reflecting the addition of the shareholder to the business.

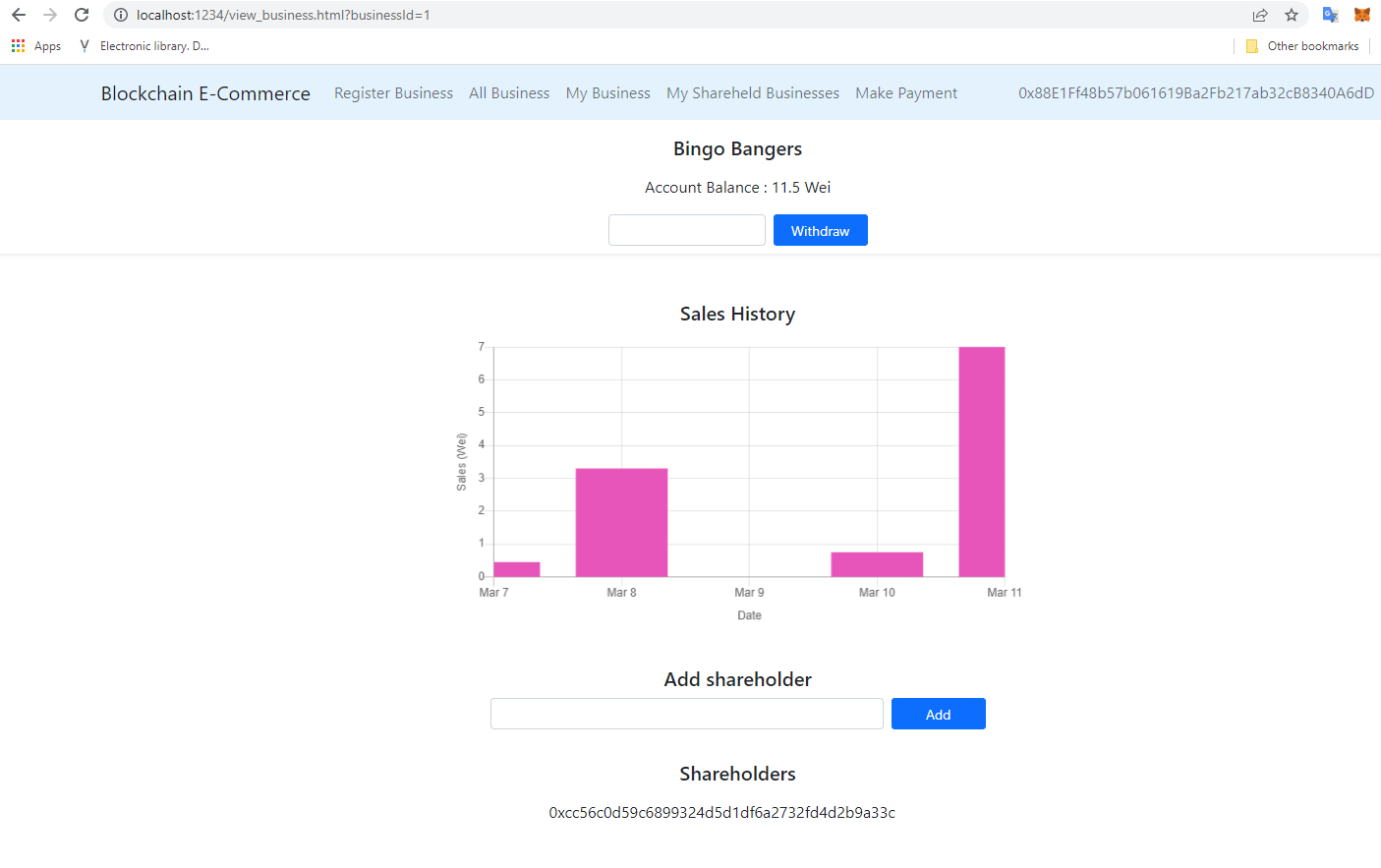


Figure :Page reflecting addition of shareholder to business

On switching the active Metamask account to that of the shareholder: 0xCC56C0d59C6899324D5d1Df6a2732fd4d2B9a33C and navigating to the “my\_shareheld\_businesses.html page”, the user found that the account added under his shareholding in the previous step was visible.

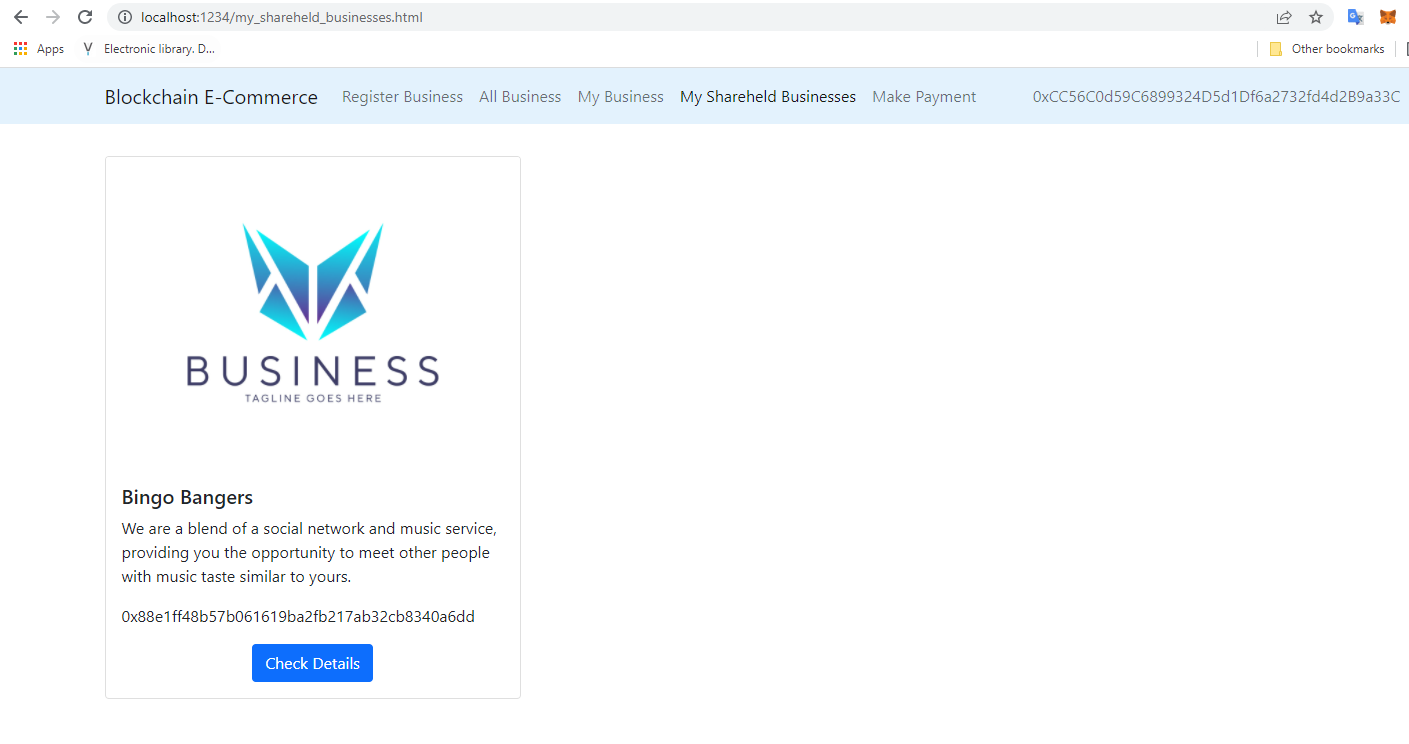


Figure :Shareholder viewing his/her businesses

Upon opening the business, the shareholder was able to access the transaction history and balance for the business. The shareholder was, however, restricted from adding other shareholders unlike the merchant who had authorization. In addition to that, the shareholder didn’t have access to the ability to withdraw ether from the business to his/her wallet.

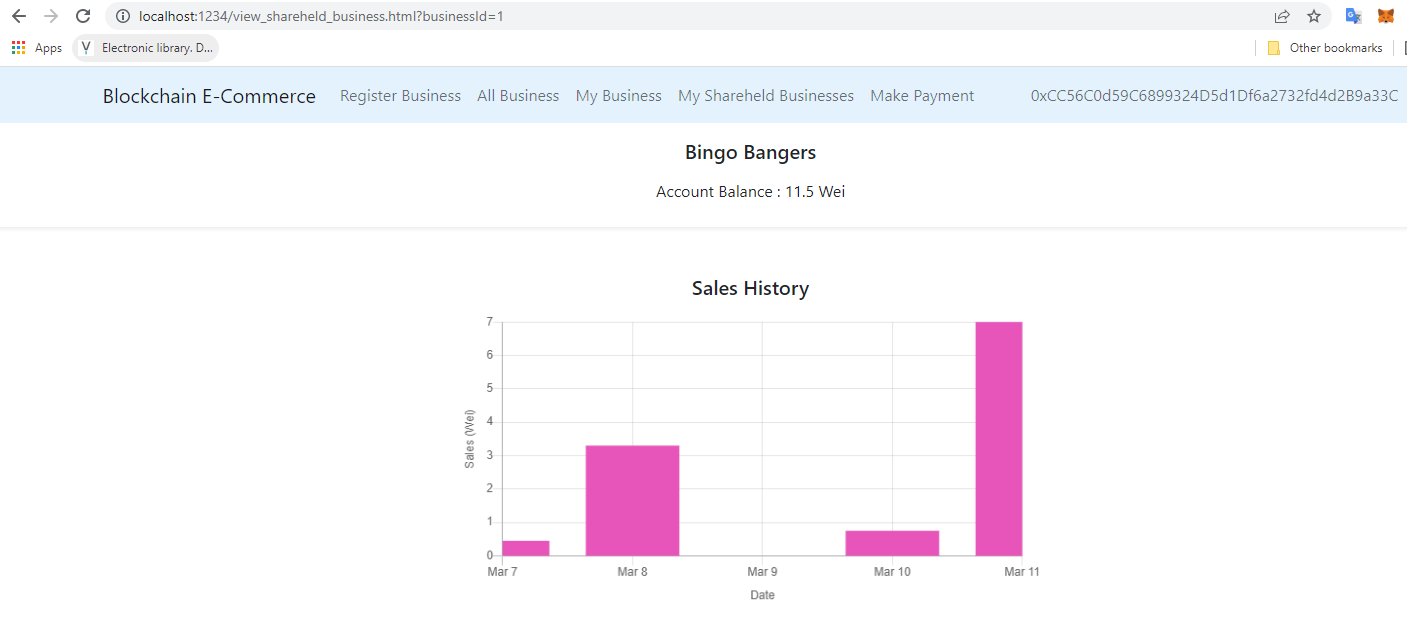


Figure :Shareholder viewing transaction history and balance of business

On switching back to account 0x88E1Ff48b57b061619Ba2Fb217ab32cB8340A6dD that owned Bingo Bangers and opening the detailed page for the business, the merchant was able to withdraw ether from the account. The business account balance and Metamask wallet balance of the merchant before withdrawing 2.4 Wei are shown

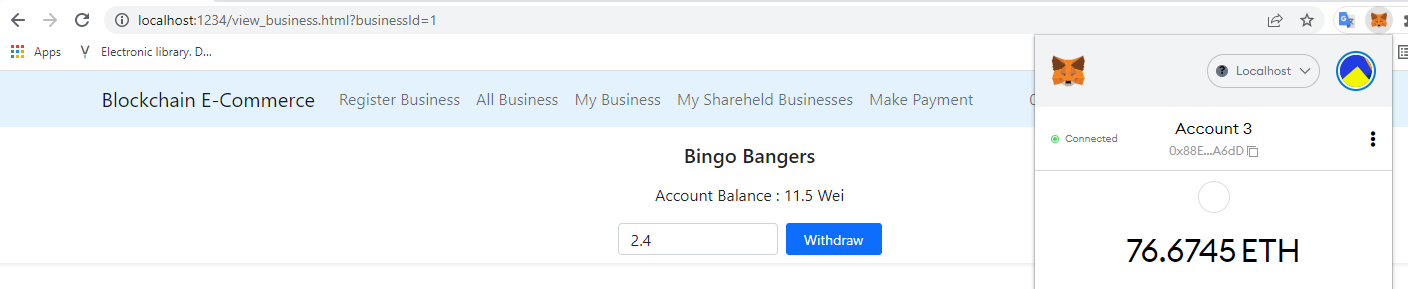


Figure :Business and wallet balance before withdrawing ether from business

The business account balance after withdrawing 2.4 Wei to the merchant’s Metamask wallet are shown reflecting this transaction. The new business balance held by the smart contract was 9.1 Wei from 11.5 Wei. The merchants wallet had increased from 76.6745 ether to 79.0738 ether.

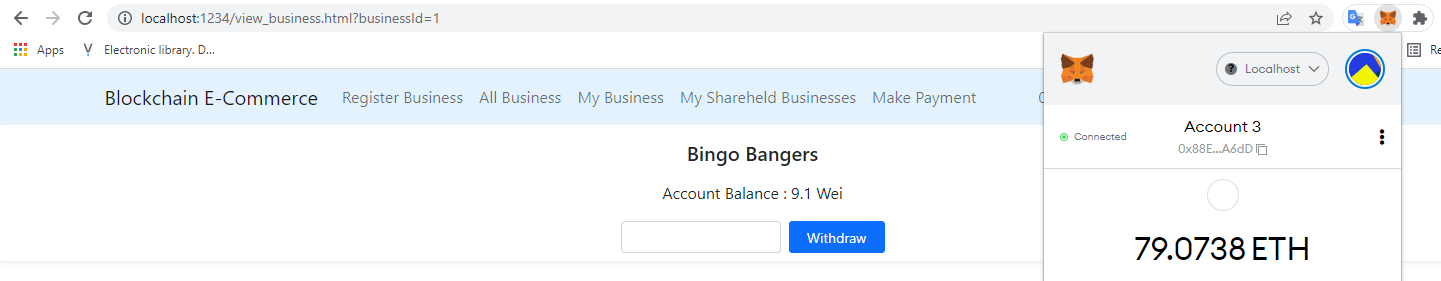


Figure :Business and wallet balance after withdrawal from the business

# Evaluation of the project

The aim of this chapter is to evaluate the results of the project in relation to the original project’s requirements.

## Gateway for payments to businesses

The smart contract exposed a public function that enabled transfer of ether from the external environment to a business registered on the platform. The function took the business ID, transaction ID and date of transaction as parameters. The implication of this was that any Dapp could call this function – making the smart contract services integratable to existing e-commerce platforms.

As a demonstration, the project built an interface to simulate payment of ether to a business registered to the platform. The interface required the user to key in the business ID to transfer ether to, a transaction ID and amount to transfer. It was observed that with Metamask installed on the user’s browser, ether could easily be transferred from a customer to the business using the service using simple calls.

In addition to making payments, the platform allowed ether to be held in escrow by the smart contract, and keep account of this, until the owner of the business decided to withdraw the money. The UI developed to accomplish this allowed the merchant to specify the amount of ether to withdraw out of the balance held by the smart contract for the business. This amount was then deducted from the business’s balance and transferred to the merchant’s Metamask wallet.

## Storing of transactions

Mappings played an important role in the storage of transactions made to businesses via the smart contract. The mapping structure developed to store this information was a state variable – meaning that the data stored there was immutable. For the tests made on the local network, it was observed that payments made persisted even after the browser was refreshed. Provided Ganache was running to provide the EVM, and the smart contract deployed, the data so stored could not be altered.

Transactions were stored in an intuitive two-level mapping nest. The key of the outer mapping was the business ID, with the value being an inner mapping. The key to the inner mapping was a transaction ID generated for internal use in the smart contract, with the value being a structure that enabled storage of the date and amount of ether transferred. This simple and intuitive design allowed transactions to be quickly inserted into the state variable containing purchases.

## Display of transactions

The platform built a user interface that enabled a shareholder or the merchant to a particular business visually track the chronological series of transactions made to the business. A bar chart was used to this regard to display the daily total sales made to the business over time.

Whereas this graphical tool provided a quick overview into the performance of the business, it was noted that this aspect of the platform was less developed compared to others. It would have been an extra advantage to the use cases of the system including analytics of the transactions data. Analytics features such as highlighting the best performing and least performing days, trends on the transactions made to the businesses, and forecasting features would have provided extra value to the merchants and shareholders.

## Usability

Usability of the system was tested using cognitive walkthrough. The cognitive walkthrough for this project is attached in appendix item 1.

From the cognitive walkthrough conducted, it was found that users were able to easily navigate to and register a business with the platform. It was. However, noted that the prompt showing successful creation of a new business was not sufficient enough to provide extra direction on how to access the user’s business page.

It was found that checking of all businesses registered to the platform was easy and intuitive from the menu. Users were noted to spend some time in order to navigate to their business page. The distinction between the page for viewing all businesses registered to the platform and those belonging to the user wasn’t clear enough.

It was noted that merchants found it easy and straight-forward to withdraw ether from the business’s account. The deduction of the account’s balance was easy to understand on withdrawing. However, it was found that user’s who weren’t familiar with Metamask found it trickier to check their wallet’s balance increment.

Viewing of transaction history was found to be easy for the users. The merchants and shareholders found it intuitive interpreting the bar chart depicting the transactions made to the business.

The addition of shareholders to a business by merchants had split opinions. Tech savvy users who were familiar with Metamask found it easy to input the shareholders’ account and adding them to their business. Other users found it difficult adding shareholders to their business. The input field used in the platform for addition of this and instructions provided to do so weren’t intuitive enough for these users.

## Security

Furthermore, access control mechanisms were defined in the smart contract to limit access of information only to authorized partied. The scope of the state variable defined to store purchases was limited only to the smart contract (was not public). This meant that the only access point to the mapping was through the getter functions defined by the smart contract. This was deemed important to preventing anyone from willy-nilly inspecting a business’s transactions – an avenue that could be exploited for espionage among other ill intents.

The getter functions for accessing the purchases information had modifiers to enforce access control. Only the registered shareholders to a business or its owner could access the purchases information related to that business. As such, unauthorized users could not access the transaction history and balance of a business.

Withdrawal of funds held in escrow on the smart contract for a given business could only be done by the merchant owner that registered the business. This access control mechanism limited even shareholders who had access to the business’s books from making withdrawals. A central point of withdrawal of ether from the smart contract from the business was deemed necessary to prevent mismanagement or embezzlement of the business’s cash.

# Recommendations

It was noted that the project left room for some areas to be built upon and modified in future works.

The UX and usability of the platform should be investigated further. The cognitive walkthrough conducted in this project to this regard was limiting in terms of sample size and representation. The aspects of the UI that were not user friendly or intuitive could be revised. Additional methods could be used in the review of the usability of the platform in order to refine the user experience on the platform to facilitate smooth operation.

Data analytics could be applied to the transaction data stored on the smart contract. This would provide additional value to the merchants and shareholders. Additional visualization tools could be used to provide deeper insight into the transactions by the platform’s users. Predictive tools could be used to find trends in the transactions handled by the business, and provide forecast into the projected transactions into the future. This would also help the platform’s users obtain business-actionable intelligence from this data.

Penetration testing could be conducted on the smart contract to expose any vulnerabilities that may be as a result of the code so written. This would be critically important to ensuring that the smart contract doesn’t leave vulnerabilities capable of being exploited by malicious parties, especially considering that such a system has the potential to handle thousands of transactions, and being accountable to a multitude of businesses.

# Appendix

**Cognitive Walkthrough Script**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cognitive Walkthrough for Platform Usability Testing** | | | | |
|  | Will the user try to achieve the right effect? | Will the user notice that the correct action is available? | Will the user associate the correct action with the effect that the user is trying to achieve? | Will the user see that progress is being made toward the solution of the task? |
| Task 1: Registering a new business |  |  |  |  |
| Identify where to click to access area for registering business |  |  |  |  |
| Fill the relevant business details |  |  |  |  |
| Know where to click to register business |  |  |  |  |
| Realize the business was registered |  |  |  |  |
| Task 2: Checking businesses registered to the platform under the user |  |  |  |  |
| Identify where to click to access area for checking user’s businesses |  |  |  |  |
| Task 3: Registering a new shareholder to the user’s business |  |  |  |  |
| Navigate to business’s page |  |  |  |  |
| Fill shareholder’s details to be added to the business |  |  |  |  |
| Identify where to click to register this shareholder to the business |  |  |  |  |
| Confirm if the shareholder has been added to the business |  |  |  |  |
| Task 4: Withdrawing ether from business account |  |  |  |  |
| Navigate to business’s page |  |  |  |  |
| Check the ether balance of the business on the platform |  |  |  |  |
| Fill in amount to withdraw |  |  |  |  |
| Identify where to click to withdraw ether |  |  |  |  |
| Confirm if transfer has been complete |  |  |  |  |
| Task 5: Check business transaction history |  |  |  |  |
| Navigate the business’s page |  |  |  |  |
| Realize view showing transaction history of the business |  |  |  |  |