



**NEW HORIZON**  
**COLLEGE OF ENGINEERING**

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC  
Accredited by NAAC with 'A' Grade, Accredited by NBA

**A PROJECT REPORT  
ON**

**“ETHEREUM DECENTRALISED BANKING APPLICATION”**

Submitted in partial fulfillment for the award of the degree of

**BACHELOR OF ENGINEERING  
IN  
COMPUTER SCIENCE AND ENGINEERING**

**BY**

**MADHUMITHA R (1NH16CS745)**

**Under the guidance of**

**Ms. R Jaya**

Sr. Assistant Professor,

Dept. of CSE

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**(Autonomous Institution Affiliated to VTU & Approved by AICTE)**  
**Accredited by NAAC 'A', Accredited by NBA**

Outer Ring Road, Panathur Post, Kadubisanahalli,

Bangalore – 560103

# ABSTRACT

The objective of the Ethereum Decentralized Banking Application is to design and develop a secure online Banking Application using Blockchain.

Some customers avoid online banking as they perceive it as being too vulnerable to fraud. The security measures employed by most banks are never 100% safe, but in practice the number of fraud victims due to online banking is very small. Indeed, conventional banking practices may be more prone to abuse by fraudsters than online banking. Credit card fraud, signature forgery and identity theft are far more widespread “offline” crimes than malicious hacking. Bank transactions are generally traceable and criminal penalties for bank fraud are high. Online banking can be more insecure if users are careless, gullible or computer illiterate. An increasingly popular criminal practice to gain access to a user’s finances is phishing, whereby the user is in some way persuaded to hand over their password(s) to the fraudster.

Building applications on Ethereum is pretty similar to a regular application calling a backend service. The hardest part is writing a robust and complete smart contract.

Since a blockchain is a permanent record of transactions that are distributed, every transaction in the banking application can irrefutably be traced back to exactly when and where it happened without revealing the user’s identity. Past deposits and withdrawals cannot be changed, while the present can’t be hacked, because every transaction is verified by every single node in the network.

# ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing my deep sense of gratitude to **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha** , Principal NHCE, for his constant support and encouragement.

I am grateful to **Dr.Prashanth C.S.R**, Dean Academics, for his unfailing encouragement and suggestions, given to me in the course of my project work.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I express my gratitude to **Ms. Jaya R**, Senior Assistant Professor, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

Finally a note of thanks to the teaching and non-teaching staff of Dept of Computer Science and Engineering, for their cooperation extended to me, and my friends, who helped me directly or indirectly in the course of the project work.

**MADHUMITHA R**  
**(1NH16CS745)**

## LIST OF FIGURES

Figure 2.1:	Features of Blockchain	3
Figure 2.2:	Solidity Program	4
Figure 2.3:	Compiling the Smart Contract	5
Figure 2.4:	Running the Smart Contract	6
Figure 2.5:	MetaMask Notification	6
Figure 2.6:	Deployed Contracts	7
Figure 2.7:	MetaMask Status	7
Figure 2.8:	Transaction details on Etherscan	8
Figure 2.9:	MetaMask Transaction History	8
Figure 2.10:	Installing the Truffle packages	9
Figure 2.11:	Installing the Ganache packages	9
Figure 2.12:	Ganache Accounts	10
Figure 2.13:	Truffle init	10
Figure 2.14:	Truffle Directories	11
Figure 2.15:	Contract Directory	11
Figure 2.16:	Truffle compile	11
Figure 2.17:	Migrations Directory	12
Figure 2.18:	Contents of "2_deploy_contracts.js"	12
Figure 2.19:	Contents of "truffle-config.js"	12
Figure 2.20:	Truffle migrate	13
Figure 2.21:	Successful deployment of Smart Contract	13
Figure 5.1:	Tracking deployment through console	19
Figure 5.2:	Deployed Contracts	19
Figure 5.3:	Withdraw function	20
Figure 5.4:	Tracking withdrawal and balance through console	20
Figure 5.5:	Balance after withdrawal	20
Figure 5.6:	Deposit function	21
Figure 5.7:	Tracking deposit and balance through console	21
Figure 5.8:	Balance after deposit	21

# CONTENTS

<b>ABSTRACT</b>	<b>i</b>
<b>ACKNOWLEDGEMENT</b>	<b>ii</b>
<b>LIST OF FIGURES</b>	<b>iii</b>
<b>1. INTRODUCTION</b>	
1.1. INTRODUCTION TO BLOCKCHAIN	1
1.2. OBJECTIVES OF THE PROJECT WORK	1
1.3. OUTCOMES OF THE PROJECT	2
<b>2. BLOCKCHAIN AND SMART CONTRACTS</b>	
2.1. CHARACTERISTICS OF BLOCKCHAIN	3
2.2. STEPS FOR CREATION OF A SMART CONTRACT	4
2.3. STEPS TO DEPLOY A SMART CONTRACT	5
2.4. STEPS TO DEPLOY A SMART CONTRACT WITH TRUFFLE	9
<b>3. REQUIREMENTS AND SPECIFICATIONS</b>	
3.1. HARDWARE REQUIREMENTS	14
3.2. SOFTWARE REQUIREMENTS	14
3.3. TOOLS BEING USED	14
<b>4. IMPLEMENTATION</b>	<b>16</b>
<b>5. SNAPSHOTS</b>	<b>19</b>
<b>6. CONCLUSION AND FUTURE WORK</b>	<b>22</b>
<b>REFERENCES</b>	<b>23</b>

## CHAPTER 1

### INTRODUCTION

#### 1.1. COURSE OBJECTIVES

- Understand the fundamentals of Block chain Technology.
- Apply the various cryptographic mechanisms used in Block chain.
- Design smart contracts.
- Understanding Ethereum environment and wallets.
- Install and administer Ethereum network and development environment.
- Design test cases for smart contracts.

#### 1.2. PROBLEM STATEMENT

This project mainly focuses on creating and interacting with the smart contracts within an application. Basically, we will have a Struct that describe a Simple bank system. With Structs, we will be able to assign multiple properties to them such as fromAdress, toAddress, amount and so on.

Struct definition, as the name suggests is just a definition but not a variable itself. We will create an instance of this Struct every time a new spending request is created.

The features that will be included in this project are:

- Display the balance of an account holder.
- Check if the User-entered ID is unique.
- Ask and store more information about a User.
- Add in an option to see whether a person with a specific id has deposited or not.
- Initializes with owner, anyone can deposit money but only account owner.
- Can withdraw money from the account. Also let the owner to check the balance.
- Require Users to have an Ethereum Address to withdraw or deposit.

### **1.3. OUTCOMES OF THE PROJECT**

- Since a blockchain is a permanent record of transactions (votes) that are distributed, every deposit can irrefutably be traced back to exactly when and where it happened without revealing the account holder's identity.
- Transactions are verified and approved by consensus among participants in the network, making fraud more difficult.
- The full chronologies of events that take place are tracked, allowing anyone to trace or audit prior transactions.
- Past deposit cannot be changed, while the present can't be hacked, because every transaction is verified by every single node in the network.
- Any outside or inside attacker must have control of 51% of the nodes to alter the record.

## CHAPTER 2

# BLOCKCHAIN AND SMART CONTRACTS

### 2.1. CHARACTERISTICS OF BLOCKCHAIN

#### 1. Consensus

For a transaction to be accepted and recorded on the blockchain, all the participants must agree to follow the same rules. This is the consensus. If a transaction violates one of the rules the network agreed on, the transaction will be considered invalid.

#### 2. Provenance

Participants know where the assets came from and how its ownership has changed over time. Each asset's (whatever it is, tangible, intangible, digital) provenance must be traceable.

#### 3. Immutability

No participant can modify a transaction after it has been recorded on the ledger. Doesn't matter who you are, you just do not have the power to do that. If an error occurs, a new transaction must be used to reverse the error.

#### 4. Finality

In a blockchain network, there is only one source of truth. There is only one ledger for the whole network. To know who owns what, or to study a particular transaction, there is only one place to go.

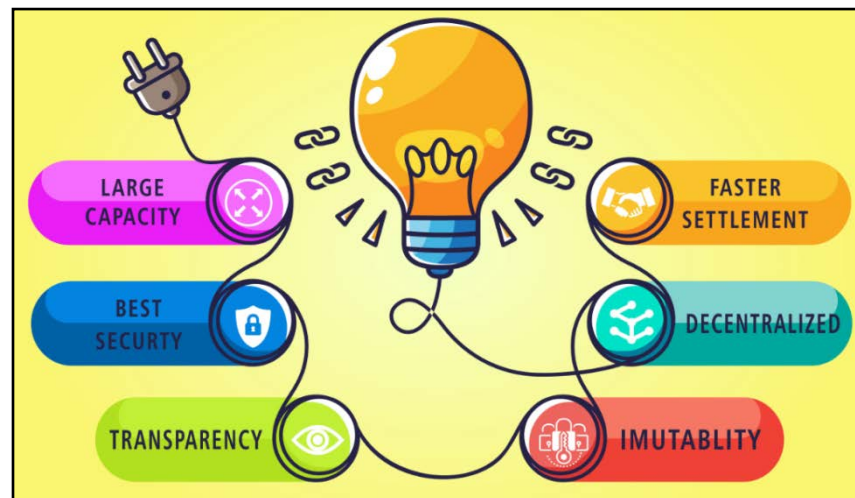


Figure 2.1: Features of Blockchain



## 2.2. STEPS FOR CREATION OF A SMART CONTRACT

1. Go to <http://remix.ethereum.org> to launch your remix IDE.
2. The Remix IDE creates a sample voting contract for you called Ballot.sol. But we want to create a simple contract of our own from scratch. Click on the + (plus) button at the top left corner to create a new file.



```
10 contract SimpleBank {
11     struct Transaction {
12         address fromAddress;
13         address toAddress;
14         uint amount;
15     }
16
17     address owner;
18     Transaction[] transactions;
19
20     modifier onlyOwner() {
21         require(owner == msg.sender);
22         _;
23     }
24
25     function SimpleBank() public {
26         /* Set the owner to the creator of this contract */
27         owner = msg.sender;
28     }
29
30     function deposit(address _fromAddress, uint _depositAmount) public {
31         /* Add the amount to the user's balance, call the event associated with a deposit,
32            then return the balance of the user */
33         Transaction memory transaction = Transaction(_fromAddress, owner, _depositAmount);
34         transactions.push(transaction);
35     }
36
37     function withdraw(address _toAddress, uint _withdrawAmount) public onlyOwner {
38         /* If the sender's balance is at least the amount they want to withdraw,
39            Subtract the amount from the sender's balance, and try to send that amount of ether
40            to the user attempting to withdraw. IF the send fails, add the amount back to the user's balance
41            return the user's balance.*/
42         require(balance() >= _withdrawAmount);
43         Transaction memory transaction = Transaction(owner, _toAddress, _withdrawAmount);
44         transactions.push(transaction);
45     }
46 }
```

Figure 2.2: Solidity Program

3. Give the file a name similar to Sample.sol.
4. Type your code in the file and save.

### 2.3. STEPS TO DEPLOY A SMART CONTRACT

5. To deploy our smart contract to the Ethereum blockchain, we need to deploy through MetaMask via the injected web3 and in our case running on the Ropsten TestNetwork.
6. Let's compile our smart contract. Click on the compile tab at the top right corner and then click on the button start to compile to compile the smart contract.
7. Hopefully it should compile without any error.

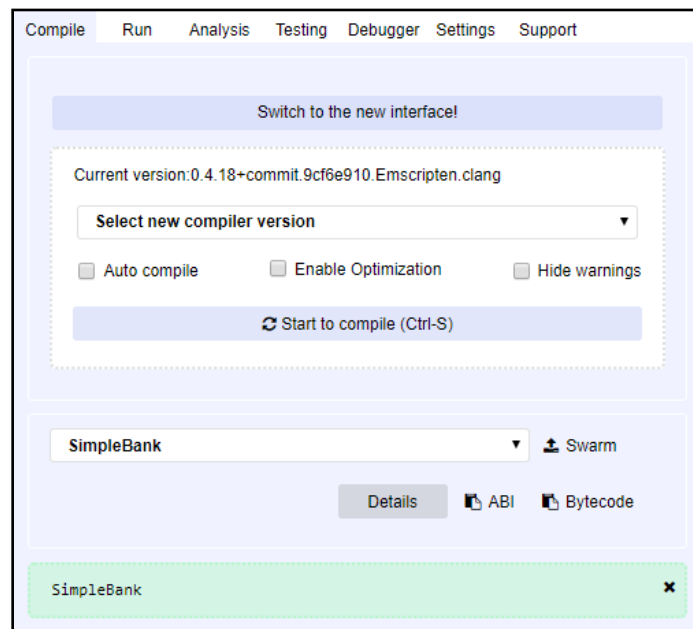


Figure 2.3: Compiling the Smart Contract

1. Next click on the Run tab to deploy the smart contract to the Ethereum blockchain.
2. Click on the Deploy button to deploy the contract to the blockchain. This action would invoke MetaMask via the Injected Web3.
3. You can increase the Gas Price to maybe 3 or 4 GWEI for faster transaction mining. Finally click on the submit button to add the contract to the blockchain.

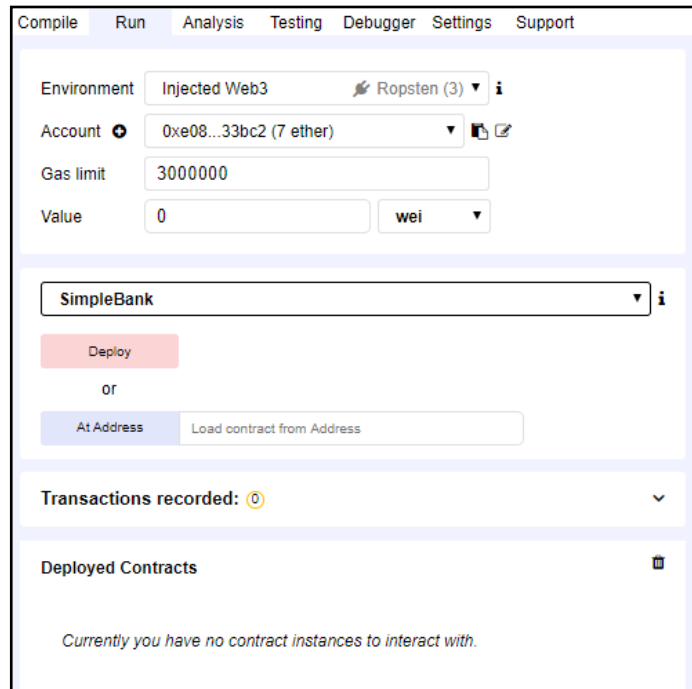


Figure 2.4: Running the Smart Contract

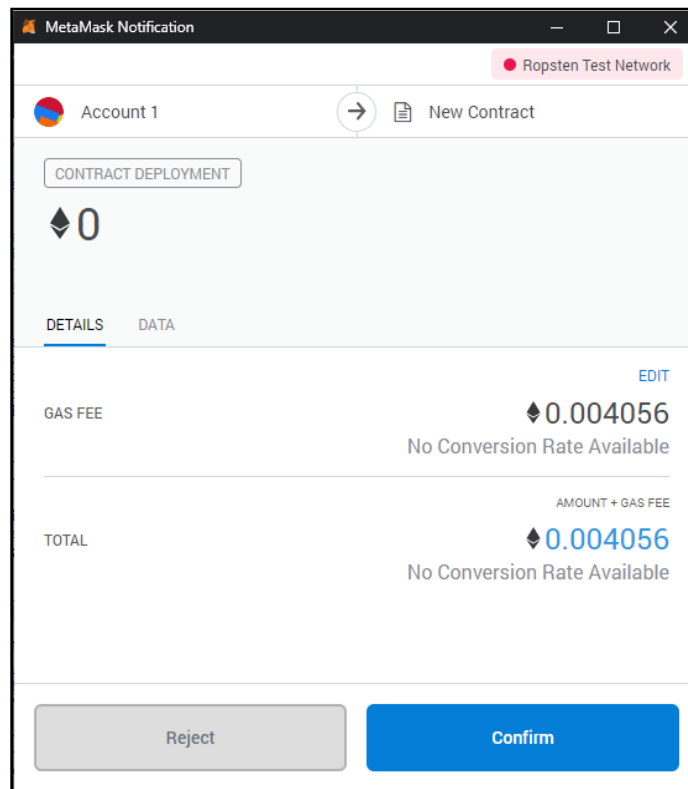


Figure 2.5: MetaMask Notification

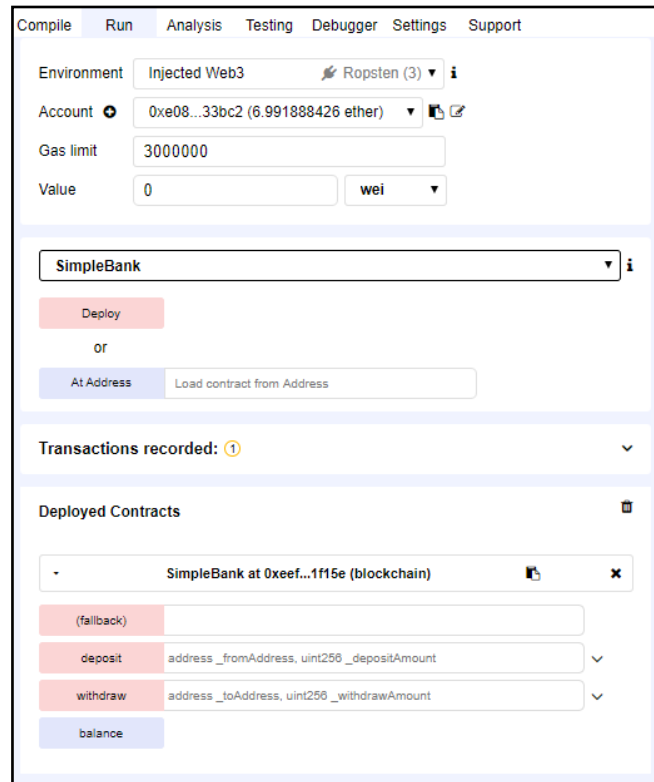


Figure 2.6: Deployed Contracts

4. After that click on the “Sent” tab on MetaMask to view your contract status.

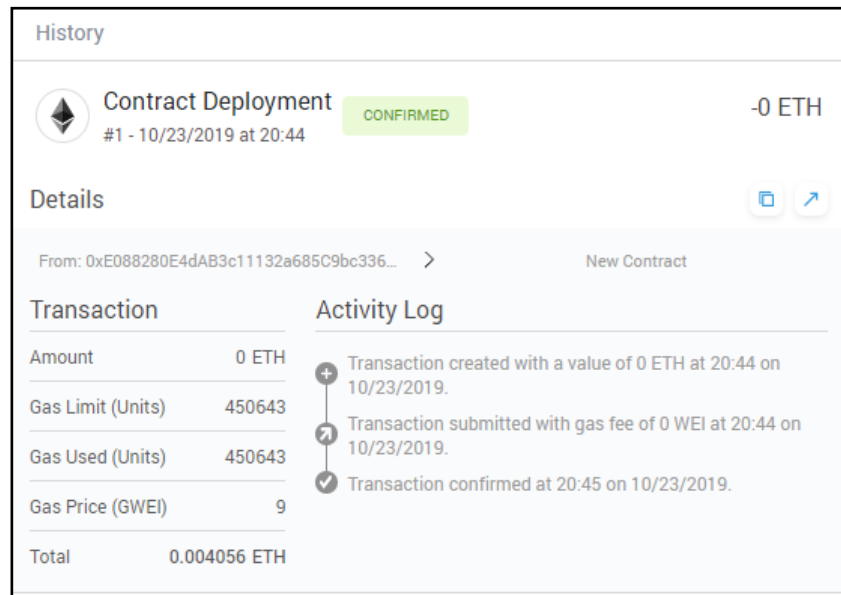


Figure 2.7: MetaMask Status

- Then click on the newly created contract to view you contract information on Etherscan.

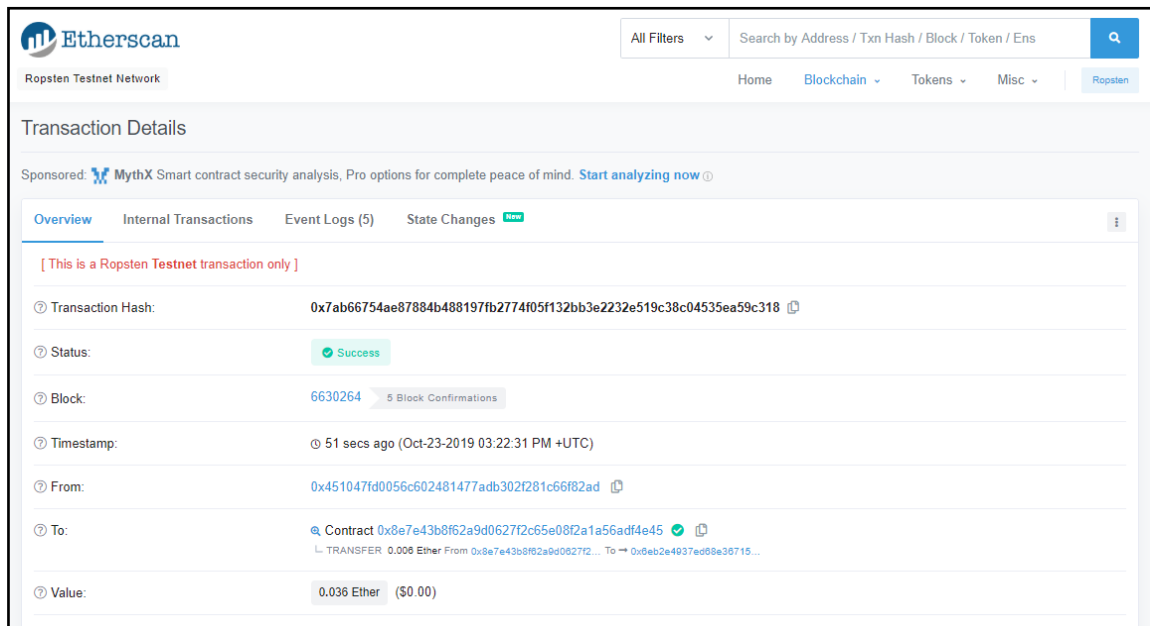


Figure 2.8: Transaction details on Etherscan

Congratulations, you have now successfully created and deployed a smart contract into the Ethereum blockchain.

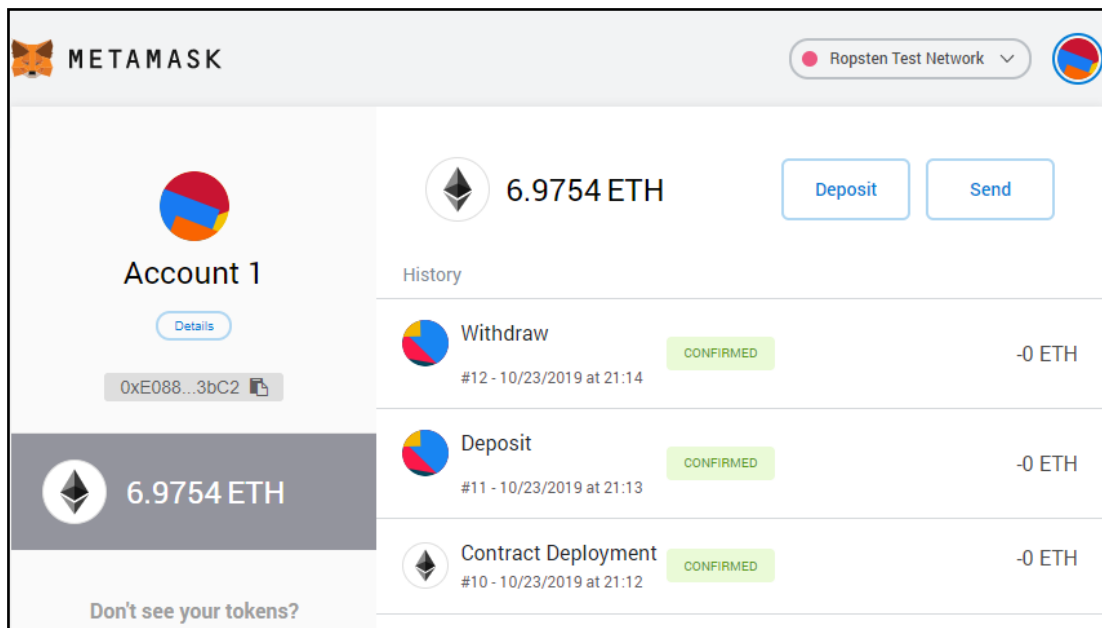


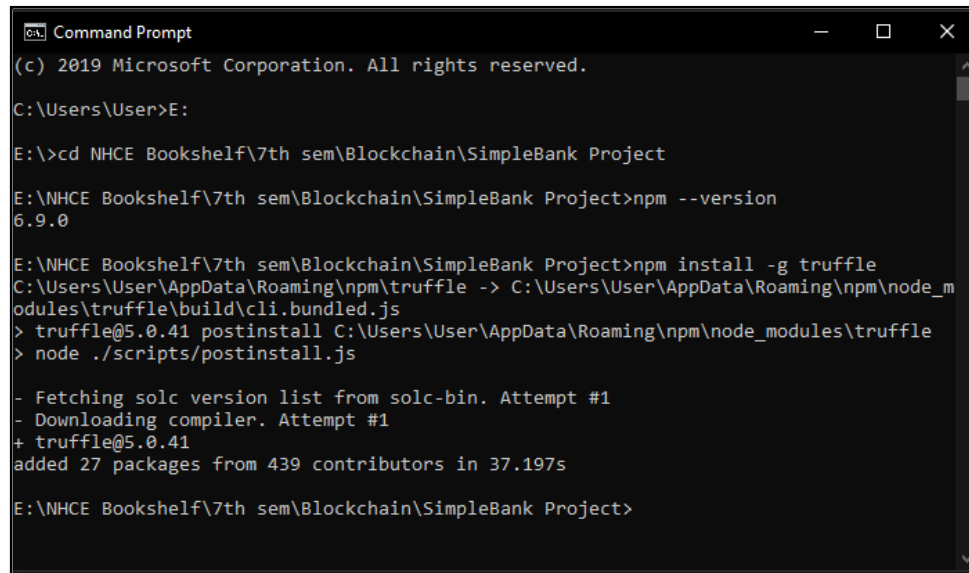
Figure 2.9: MetaMask Transaction History

## 2.4. STEPS TO DEPLOY A SMART CONTRACT WITH TRUFFLE

Truffle is basically a development environment where you could easily develop smart contracts with its built-in testing framework, smart contract compilation, deployment, interactive console, and many more features.

It is mostly recommended for developers who want to build JavaScript projects based on smart contracts (like DApps). With Truffle, you are able to have a better simulation of a real blockchain environment.

1. First, we install the packages we will be using by first opening Terminal/CMD and then execute this command:



```
Command Prompt
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\User>E:

E:\>cd NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project

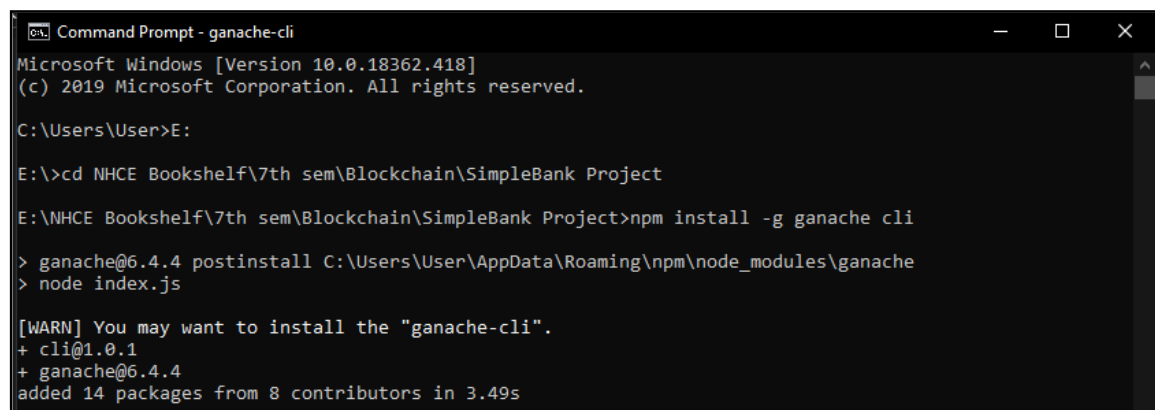
E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>npm --version
6.9.0

E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>npm install -g truffle
C:\Users\User\AppData\Roaming\npm\truffle -> C:\Users\User\AppData\Roaming\npm\node_modules\truffle\build\cli.bundled.js
> truffle@5.0.41 postinstall C:\Users\User\AppData\Roaming\npm\node_modules\truffle
> node ./scripts/postinstall.js

- Fetching solc version list from solc-bin. Attempt #1
- Downloading compiler. Attempt #1
+ truffle@5.0.41
added 27 packages from 439 contributors in 37.197s

E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>
```

Figure 2.10: Installing the Truffle packages



```
Command Prompt - ganache-cli
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\User>E:

E:\>cd NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project

E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>npm install -g ganache cli
> ganache@6.4.4 postinstall C:\Users\User\AppData\Roaming\npm\node_modules\ganache
> node index.js

[WARN] You may want to install the "ganache-cli".
+ cli@1.0.1
+ ganache@6.4.4
added 14 packages from 8 contributors in 3.49s
```

Figure 2.11: Installing the Ganache packages

- Then the ganache cmd prompt will display 10 accounts and their private keys.

```

C:\Users\User>ganache-cli
Ganache CLI v6.7.0 (ganache-core: 2.8.0)

Available Accounts
=====
(0) 0xB7e7AE46C90f585a7bE129455101ba22FcC75d59 (100 ETH)
(1) 0x1Ea6b19F089c7dAfaF5Af4355C9c7A431A00Fb8A (100 ETH)
(2) 0x3c78B747C37A2830ceb890FAD5C9CE304407b36b (100 ETH)
(3) 0x57598081700f7f3033dEba58Afb5D0eC08cA2278 (100 ETH)
(4) 0x04CD6a204F9FfCbD6d41AaCB92dFDB3666412660 (100 ETH)
(5) 0x608E771c8C6A87A78B130FfD76A7598f2aE8Fb1d (100 ETH)
(6) 0xC88D37Bc08aDd9b64b48d720cc71dd8Fa23F3853 (100 ETH)
(7) 0xA58F5847fd67aaCdB277a9EcF915844f8c5098E6 (100 ETH)
(8) 0x497AC8Cc43dAFc770ceC3a59b8810E4f90949F26 (100 ETH)
(9) 0xe5310FBe8C074a202d9B22D23937E45b7a999ef1 (100 ETH)

Private Keys
=====
(0) 0x175a7050bb5437fd99598fa81894f7aa24710ca917d1233afe1715fdda5ac281
(1) 0x581fade7b4d36d546bef62b7cbd868f0a49dc8267460e6dd1294436035d02823
(2) 0x8db5288ab7f1727666c46937537cb23a10edde30ec6bcf928f3bf89fa01d32d5
(3) 0xf83868f3ee56130bf2a3461427f90ec2cc3b745e47357c52af91118f84ed200e
(4) 0x0d0b04f4c4cbffe5de3b0e90814f5821aa618612e37dc1fad439ef44acbbbd3f
(5) 0x760841622e88ba5d5eac199a4846ec912528ed8f78677aa3e39e1c307059c015
(6) 0x8320c1a1c1080adb930f0b835654a3987f06b0087101157f1cbdab41ba936013
(7) 0x49a322d7bae82a0a5ae77f597560bebe57600eb9a475400a7274931990341dc3
(8) 0x61e4e16f08801c09e1a0fd09359caabe1b4f363b2000c2fcd3f555be92241
(9) 0x7679e4eade51a0759b9e32556256b26efdd6a865ae0f4e31594440e964f79131

HD Wallet
=====
Mnemonic:      unveil bright retire grief pass music melt submit truth elegant crop stone
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
=====
20000000000

Gas Limit
=====
>

Listening on 127.0.0.1:8545

```

Figure 2.12: Ganache Accounts

- This command creates a bare Truffle project without anything else included.

```

E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>truffle init

✓ Preparing to download
✓ Downloading
✓ Cleaning up temporary files
✓ Setting up box

Unbox successful. Sweet!

Commands:

Compile:      truffle compile
Migrate:      truffle migrate
Test contracts: truffle test

```

Figure 2.13: Truffle init

4. After doing so, you should have the following files and folders:

Name	Date modified	Type	Size
contracts	20-10-2019 11:25	File folder	
migrations	20-10-2019 11:25	File folder	
test	20-10-2019 11:25	File folder	
truffle-config	20-10-2019 11:25	JavaScript File	5 KB

Figure 2.14: Truffle Directories

- **contracts/**: Directory for Solidity contracts
  - **migrations/**: Directory for scriptable deployment
  - **test/**: Directory for test files for testing your application and contracts
  - **truffle-config.js**: Truffle configuration file
5. Now let's proceed to smart contract compilation with Truffle. Before anything else, let's create a very simple smart contract named **SimpleBank.sol** and place it inside the **contracts** folder. All smart contracts you create should be placed inside this folder.

Name	Date modified	Type	Size
Migrations.sol	20-10-2019 11:25	SOL File	1 KB
SimpleBank.sol	20-10-2019 11:31	SOL File	3 KB

Figure 2.15: Contracts Directory

6. Let's try compiling our smart contract! Execute the command:

```

Node v10.16.3

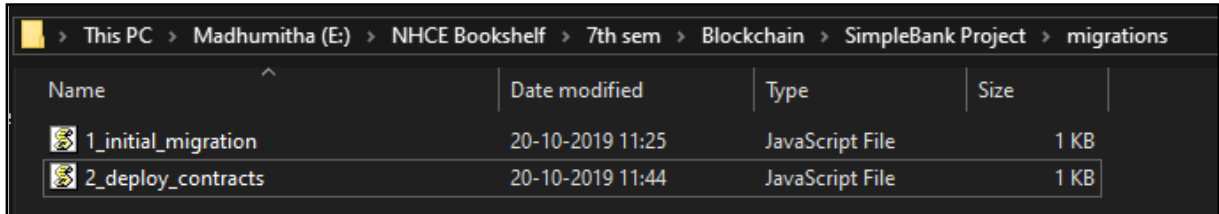
E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SimpleBank.sol
> Artifacts written to E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project\build\contracts
> Compiled successfully using:
   - solc: 0.4.18+commit.9cf6e910.Emscripten.clang
  
```

Figure 2.16: Truffle compile

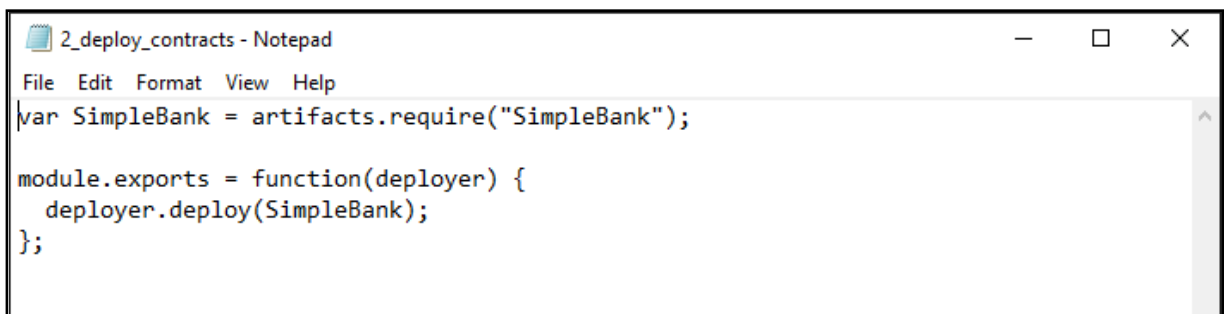


7. In order to deploy our smart contract, we'll need to create a new file in the **migrations** directory. So, create a new file named **2\_deploy\_contracts.js**, and place the following code:



Name	Date modified	Type	Size
1_initial_migration	20-10-2019 11:25	JavaScript File	1 KB
2_deploy_contracts	20-10-2019 11:44	JavaScript File	1 KB

Figure 2.17: Migrations Directory

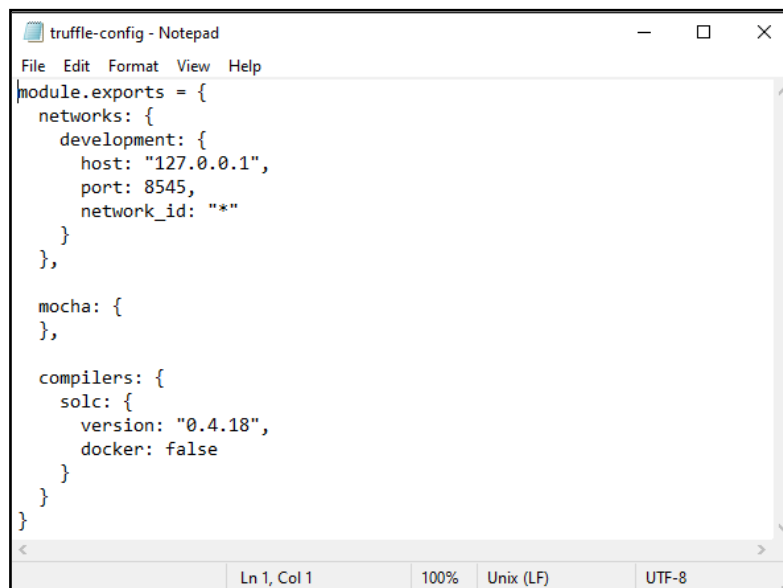


```
var SimpleBank = artifacts.require("SimpleBank");

module.exports = function(deployer) {
  deployer.deploy(SimpleBank);
};
```

Figure 2.18: Contents of "2\_deploy\_contracts.js"

8. Let's open **truffle-config.js**, and change the **networks** part to look like this:



```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*"
    }
  },
  mocha: {
  },
  compilers: {
    solc: {
      version: "0.4.18",
      docker: false
    }
  }
}
```

Figure 2.19: Contents of "truffle-config.js"

9. Let's try deploying by executing the command:

```

Command Prompt
E:\NHCE Bookshelf\7th sem\Blockchain\SimpleBank Project>truffle migrate

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
> Network name: 'development'
> Network id: 1571550896762
> Block gas limit: 0x6691b7

1_initial_migration.js
=====

Deploying 'Migrations'
-----
> transaction hash: 0xe9e98f42273007b406a85e1488c61f275292bcd58e28da5aaab78d0128f3a7
> Blocks: 0 Seconds: 0
> contract address: 0x66e7338f9594c8f8bdE7a4Eb152c59c2a25293DE
> block number: 1
> block timestamp: 1571552123
> account: 0x4367f4970eee7480ef1787C0b21ed3204dA84185
> balance: 99.99460786
> gas used: 269607
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00539214 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.00539214 ETH

```

Figure 2.20: Truffle migrate

```

Command Prompt
2_deploy_contracts.js
=====

Deploying 'SimpleBank'
-----
> transaction hash: 0xea006a117d7814bcf3b249c647a36575979c50e0fc59e4ef139df1494c42d5df
> Blocks: 0 Seconds: 0
> contract address: 0xD8602A45944281bC1f804D70087010C8960e6a33
> block number: 3
> block timestamp: 1571552124
> account: 0x4367f4970eee7480ef1787C0b21ed3204dA84185
> balance: 99.98297754
> gas used: 539535
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.0107907 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.0107907 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.01618284 ETH

```

Figure 2.21: Successful deployment of Smart Contract

## CHAPTER 3

# REQUIREMENTS AND SPECIFICATIONS

### 3.1. HARDWARE REQUIREMENTS

1. Processor : Intel core processor
2. Speed : 2.50 GHz
3. RAM : Minimum of 256MB
4. Hard disk : 1 TB

### 3.2. SOFTWARE REQUIREMENTS

1. OS : Windows 7,8,10
2. Developing language : Solidity
3. Tools used : Remix IDE, MetaMask, Etherscan

### 3.3. TOOLS BEING USED

#### 1) Remix IDE

Remix is an Ethereum IDE for the smart contract programming language called Solidity and it has an integrated debugger and testing environment.

You can go to <http://remix.ethereum.org> to launch your Remix IDE.

Remix is a good solution if you intend to:

- a) Develop smart contracts (remix integrates a solidity editor).
- b) Debug a smart contract's execution.
- c) Access the state and properties of already deployed smart contract.
- d) Debug already committed transaction.
- e) Analyse solidity code to reduce coding mistakes and to enforce best practices.
- f) Together with Mist or MetaMask (or any tool which injects web3), Remix can be used to test and debug a DApp.

## **2) MetaMask**

MetaMask is a bridge that allows you to visit the distributed web of tomorrow in your browser today. It allows you to run Ethereum dApps right in your browser without running a full Ethereum node. MetaMask includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions. You can install the MetaMask add-on in Chrome, Firefox, Opera, and the new Brave browser.

## **3) Etherscan**

Etherscan is the leading Block Explorer for the Ethereum Blockchain. A Block Explorer is basically a search engine that allows users to easily lookup confirm and validate transactions that have taken place on the Ethereum Blockchain. We are independently operated and developed by a team of individuals who are truly passionate and excited about the kinds of decentralized information and infrastructure applications that Ethereum makes possible.

You can view all your wallet transactions on <https://etherscan.io/Etherscan website>.

## CHAPTER 4

### IMPLEMENTATION

```
pragma solidity ^0.4.18;

/*
 * A simple banking contract
 * Initializes with owner, anyone can deposit money but only account owner
 * can withdraw money from the account. Also let the owner to check the balance.
 * 0x345ca3e014aaf5dca488057592ee47305d9b3e10
 */

contract SimpleBank {
    struct Transaction {
        address fromAddress;
        address toAddress;
        uint amount;
    }
    address owner;
    Transaction[] transactions;
    modifier onlyOwner() {
        require(owner == msg.sender);
        _;
    }
    function SimpleBank() public {
        /* Set the owner to the creator of this contract */
        owner = msg.sender;
    }
}
```

```
function deposit(address _fromAddress, uint _depositAmount) public {  
    /* Add the amount to the user's balance, call the event associated with a deposit,  
       then return the balance of the user */  
    Transaction memory transaction = Transaction(_fromAddress, owner, _depositAmount);  
    transactions.push(transaction);  
}
```

```
function withdraw(address _toAddress, uint _withdrawAmount) public onlyOwner {  
    /* If the sender's balance is at least the amount they want to withdraw,  
       Subtract the amount from the sender's balance, and try to send that amount of ether  
       to the user attempting to withdraw.  
       IF the send fails, add the amount back to the user's balance and return the user's  
       balance.*/  
    require(balance() >= _withdrawAmount);  
    Transaction memory transaction = Transaction(owner, _toAddress, _withdrawAmount);  
    transactions.push(transaction);  
}
```

```
function balance() public view returns (uint) {  
    uint paidIn = 0;  
    uint paidOut = 0;  
    /* Get the balance of the sender of this transaction */  
    for (uint i = 0; i < transactions.length; i++) {  
        if (transactions[i].toAddress == owner) {  
            paidIn += transactions[i].amount;  
        }  
    }
```

```
    if (transactions[i].fromAddress == owner) {  
        paidOut += transactions[i].amount;  
    }  
}  
return paidIn - paidOut;  
}  
  
// Fallback function - Called if other functions don't match call or  
// sent ether without data.  
// Typically, called when invalid data is sent  
// Added so ether sent to this contract is reverted if the contract fails  
// otherwise, the sender's money is transferred to contract  
function() public {  
    revert();  
}  
}
```

## CHAPTER 5

### SNAPSHOTS



Figure 5.1: Tracking deployment through console

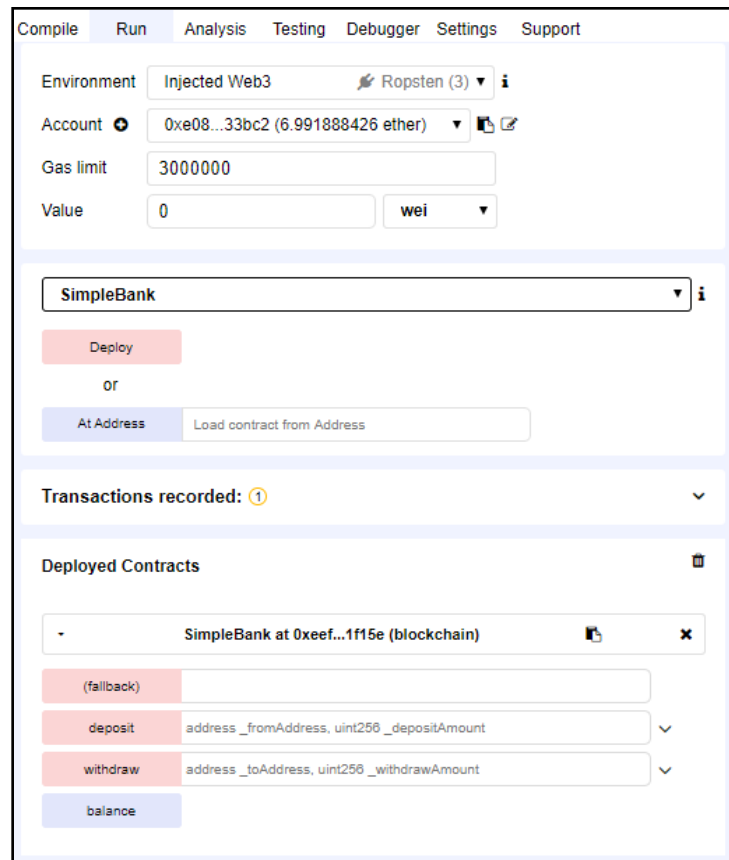


Figure 5.2: Deployed Contracts



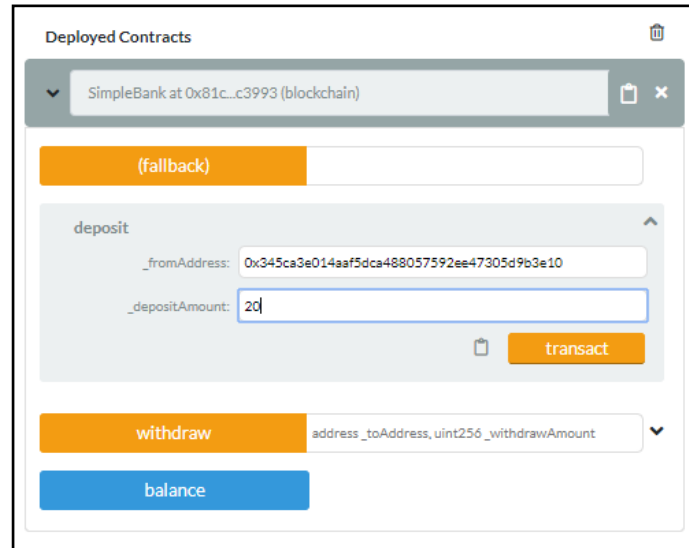


Figure 5.3: Withdraw function



Figure 5.4: Tracking withdrawal and balance through console

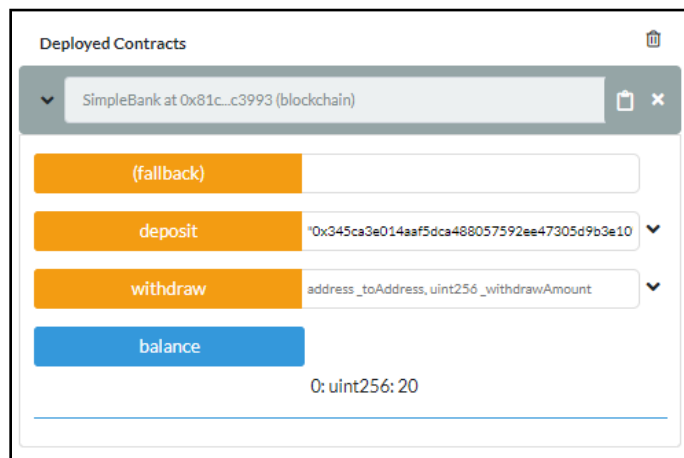


Figure 5.5: Balance after withdrawal

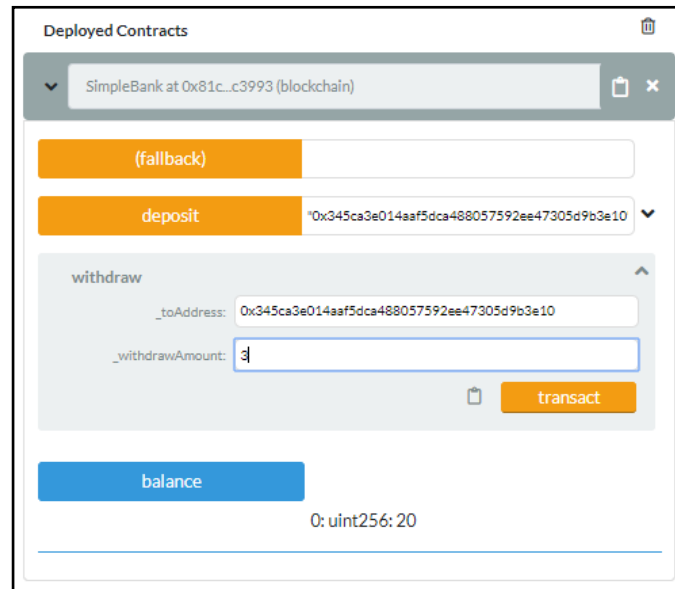


Figure 5.6: Deposit function



Figure 5.7: Tracking deposit and balance through console

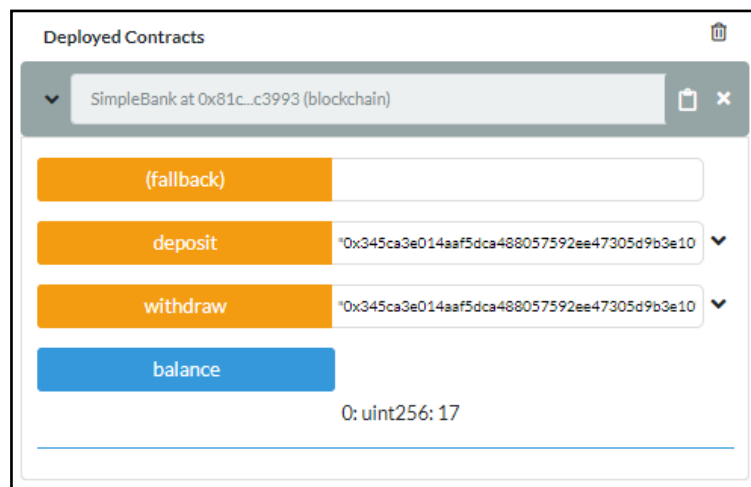


Figure 5.8: Balance after deposit

## CHAPTER 6

# CONCLUSION AND FUTURE WORK

Building applications on Ethereum is pretty similar to a regular application calling a backend service. The hardest part is writing a robust and complete smart contract.

Since a blockchain is a permanent record of transactions that are distributed, every transaction in the banking application can irrefutably be traced back to exactly when and where it happened without revealing the user's identity. Past deposits and withdrawals cannot be changed, while the present can't be hacked, because every transaction is verified by every single node in the network.

### 6.1. FUTURE WORK

This method of transaction can be used for directly depositing and withdrawing from any place instead of actually going to the banks. The main objective of the proposed system is to increase the usage of banks and even to improve the security of banking system with valid transactions.

This system will be very secure as the votes cannot be tampered because:

1. Any outside or inside attacker must have control of 51% of the nodes to alter the record.
2. Even if the attacker is able to achieve that while incorrectly entering user votes with their real IDs under the radar, end to end voting systems could allow voters to verify whether their vote was correctly entered in the system, making the system extremely safe.

## REFERENCES

- [1] Mastering Bitcoin: Programming the Open Blockchain
- [2] <https://solidity.readthedocs.io/en/v0.5.12/>
- [3] <https://www.blockchain.com/>
- [4] <https://en.wikipedia.org/wiki/Blockchain>
- [5] <https://medium.com/openberry/deploying-smart-contracts-with-truffle-1c056b452cde>
- [6] <https://www.c-sharpcorner.com/article/how-to-deploy-and-test-your-smart-contracts-on-ropsten-testnet-using-metamask-an/>