Banking Decentralized Application using Smart Contracts on Blockchain

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech.

in

COMPUTER SCIENCE & ENGINEERING

By

THIVYAVIGNESH R G (106114071)

SHARATH R (106114086)

VENKATESHWARAN K (106114103)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY TIRUCHIRAPPALLI-620015

MAY 2018

BONAFIDE CERTIFICATE

This is to certify that the project titled **Banking Decentralized Application using Smart Contracts on Blockchain** is a bonafide record of the work done by

THIVYAVIGNESH R G (106114071)

SHARATH R (106114086)

VENKATESHWARAN K (106114103)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI**, during the year 2017-2018.

DR. KUNWAR SINGH	**DR. R. LEELA VELUSAMY**
Project Guide	Head of the Department

Project Viva-voce held on _____

Internal Examiner **External Examiner**

ABSTRACT

The most shocking turn of events recently was the discovery of the fraudulent activities taking place in the Punjab National Bank. It was the result of numerous systemic failures to detect simple human malfeasance. These failures could have been easily spotted and prevented with a fraud mitigation blockchain platform. It's surprisingly common for the information settlement mechanism like SWIFT to be on a separate ledger from the payment settlement mechanism. If a bank uses a distributed-ledger platform that accommodates information settlement, then the payments and all of the associated information are available to all of the participants in the transactions,regulators and auditors.

Our project is a web based Banking Decentralised Application (DApp) built on Ethereum blockchain which focuses on preventing such fraudulent attacks on sanctioning of Loans and Tenders by decentralising the processes. The security aspects concerning the user identity authentication, bank official authentication and multi level verification of details are implemented by incorporating the notion of Public Key Infrastructure (PKI).

The Smart contracts are deployed in the backend, which runs on ethereum simulator Ganache and Ropsten testnet. The front end is developed using Javascript and HTML. The integration of components is done using ethereum wallet Metamask. This report explains the working of Banking DApp with demonstrations and code explanations accompanied by a brief elucidation on the concepts of Blockchain and Ethereum.

Keywords: Blockchain, Ethereum, Smart Contracts, Banking, DApp, Solidity, Metamask, Ganache, Ropsten

ACKNOWLEDGEMENT

We would like to thank the following people for their support and guidance without whom the completion of this project in fruition would not be possible.

DR. KUNWAR SINGH, Assistant Professor, Department of Computer Science & Engineering, our project guide, for helping us and guiding us through the course of this project with constant encouragement.

DR. (Mrs.) R. LEELA VELUSAMY, the Head of the Department, Department of Computer Science & Engineering for her inspiration and support.

Our internal reviewers, **DR. R. MOHAN**, Assistant Professor, and **DR. A. SANTHANA VIJAYAN**, Assistant Professor, Department of Computer Science & Engineering for their insight and advice provided during the review sessions.

We would also like to extend our gratitude to our individual parents and friends for their constant support.

THIVYAVIGNESH R G (106114071)

SHARATH R (106114086)

VENKATESHWARAN K (106114103)

TABLE OF CONTENTS

Title **Page No**

ABSTRACT . . . . . **ii**

ACKNOWLEDGEMENT . . . . . **iii**

TABLE OF CONTENTS . . . . . **iv**

LIST OF TABLES . . . . . **vii**

LIST OF FIGURES . . . . . **viii**

1 Blockchain **1**

1.1 Introduction 1

1.2 Structure 2

1.3 Main Characteristics 3

1.4 Consensus Models 3

1.4.1 Proof of Work 5

1.4.2 Proof of Stake 6

1.4.3 Proof of Authority 6

2 Ethereum **7**

2.1 Introduction 7

2.2 Terminologies 7

2.2.1 Address 7

2.2.2 Ethereum VM 7

2.2.3 Gas 8

2.2.4 Ether . . . 9
2.3 Ethereum vs Bitcoin . . . 9
2.4 Testnets . . . 10
2.4.1 Public Test . . . 10
2.4.2 Private Test . . . 11
2.4.3 TestRPC . . . 12
2.5 Smart Contracts . . . 12
2.6 Ethereum Wallet . . . 14

3 Ethereum DApp . . . 15
3.1 Web 2.0 vs Web 3.0 . . . 15
3.1.1 Architecture of Ethereum . . . 15
3.2 Ethereum Client . . . 16
3.2.1 Geth . . . 16
3.3 Web3 Javascript API . . . 17
3.4 Frameworks . . . 18
3.4.1 Remix . . . 18
3.4.2 Ethereum Simulators . . . 18
3.4.2.1 Ganache . . . 19
3.4.3 Mocha . . . 19
3.4.4 Chai . . . 19
3.4.5 Metamask . . . 19
3.5 Solidity . . . 19
3.5.1 Variables . . . 20
3.5.2 Functions . . . 20
3.5.3 Arrays . . . 20
3.5.4 Structs . . . 20
3.5.5 Mappings . . . 21
3.5.6 Contract Deployment . . . 21

4 Banking Decentralized Application **22**

4.1 Objective 22

4.2 Key Features 22

4.3 Implementation Approach 24

4.3.1 Ganache-CLI 24

4.3.2 Ropsten Testnet 32

5 Conclusion **35**

5.1 Technical Limitations of Block Chain 35

5.2 Conclusion 36

References **37**

Appendices **38**

A Solidity Codes **38**

B Web3 and Javascript Codes **41**

List of Tables

3.1 Web 2.0 vs Web 3.0 15
3.2 Web3 18
3.3 Management 18

List of Figures

1.1 Blockchain 1

1.2 Consensus 5

2.1 Private Testnet 12

2.2 Working of Smart Contract 13

2.3 Wallet 14

3.1 Architecture of Ethereum 16

3.2 Ethereum Client Architecture 17

3.3 Web3J 17

4.1 Starting Ganache 24

4.2 Connecting Ganache with Remix 24

4.3 Bank Database using XAMPP Phpmyadmin 25

4.4 Sign Up & Login Page 25

4.5 Bank Central Database after Sign Up by user 26

4.6 User Welcome Page 26

4.7 Message Signing by user 26

4.8 Bank Database after updation of user's signature 27

4.9 Authenticator Welcome Page 27

4.10 User Details page 28

4.11 Authenticator Loan/Tender Page 28

4.12 Invalid Authentication 28

4.13 Authenticator enters Loan details 29

4.14 Save Block Hash of Loan Details 29

4.15 Bank Database after updation of BlockHash 30
4.16 Loan details corresponding to Loan Id 30
4.17 Authenticator enters Tender details 30
4.18 Displaying Tender Winner 31
4.19 The smart contract using Remix IDE 31
4.20 Multiple Verifications using multi block hash authentications 32
4.21 Metamask and Remix Integrated 32
4.22 On updating the Loanee details to Ropsten by Authenticator 33
4.23 Processing the Transaction Request 33
4.24 Loanee Details Added Successfully 33
4.25 Transaction Information 34
4.26 Block Information 34

Chapter 1

Blockchain

1.1 Introduction

By allowing the digital information to be distributed but not copied, Blockchain technology [1] [2] has created the backbone of a new type of internet. Picture a spreadsheet that is duplicated thousands of times across a network of computers. Then imagine that this network is designed to regularly update this spreadsheet and you have a basic understanding of the blockchain.

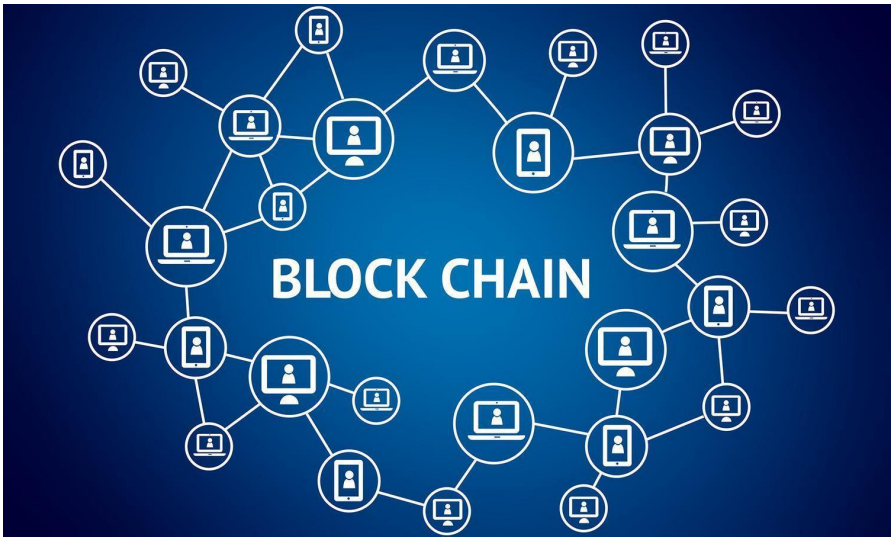

Figure 1.1: Blockchain

For use as a distributed ledger as shown in 1.1[1], a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Once recorded, the data in any given block cannot

[1] https://medium.com/@fidelvti/blockchain-for-dummies-df780e083189

be altered retroactively without the alteration of all subsequent blocks, which requires collusion of the network majority. The distributed part comes into play when sharing involves a number of people.Imagine the number of legal documents that should be used that way. Instead of passing them to each other, losing track of versions, and not being in sync with the other version, why can't all business documents become shared instead of transferred back and forth? So many types of legal contracts would be ideal for that kind of workflow.You don't need a blockchain to share documents, but the shared documents analogy is a powerful one.

1.2 Structure

Blockchains can create trust in digital data. When information has been written into a blockchain database, it's nearly impossible to remove or change it.

Block: A list of transactions recorded into a ledger over a given period. The size, period, and triggering event for blocks is different for every blockchain. Not all blockchains are recording and securing a record of the movement of their cryptocurrency as their primary objective.

Chain: A hash that links one block to another, mathematically "chaining" them together. This is one of the most difficult concepts in blockchain to comprehend. It's also the magic that glues blockchains together and allows them to create mathematical trust.

Network: The network is composed of "full nodes." Think of them as the computer running an algorithm that is securing the network. Each node contains a complete record of all the transactions that were ever recorded in that blockchain.

To understand the blockchain technology structure, think about layers in a geological formation. With seasons, the surface layer might change. The surface layer can also be blown away before it has time to settle. However, when you go several inches deep, the layers became more and more stable. When you look a hundred feet down, you will see rocks that have remained undisturbed for centuries. In the same way, in blockchain, the recent blocks might be changed easily. But once you go deep into the blockchain, blocks are less and less likely to change. Beyond 100 blocks, there is so much perma-

nency. While the likelihood of any block being changed always exists, the possibility of such an occurrence decreases as time passes until it becomes insignificant.

1.3 Main Characteristics

Distributed : When we say Blockchain is distributed it means it can run on any computer provided by volunteers across the globe. There is no central database, no authority to track it, that also says it cannot be shut down or hacked. Any digital transaction can happen directly between two parties, no intermediaries are required.

Encrypted : Gone are the days of firewalls, each block of blockchain is heavily encrypted that involves public and private keys. The encryption helps to secure and to maintain the uniqueness of each block.

Public : Well, to some extent blockchain is public in nature, as anyone in the network can view it. That makes the transactions transparent and open; there is no hiding of transactions.

Inclusive : Could you imagine an international transaction happening without documents/legal channels? Well, Satoshi did imagine a simplified payment verification mode that could work on a mobile device and does not need legal documents.

Immutable : Once a transaction has been initiated, verified and validated by others, it is added as a new block with the timestamp and linking to preceding block with a unique signature. It cannot be changed or altered and remains in the ledger permanently.

Historical : You cannot steal a block or for that matter bitcoin. Just for example the units of bitcoin that are present if taken by someone, then he needs to rewrite its history entirely. That makes it impossible for someone to steal the bitcoins making them historical.

1.4 Consensus Models

Blockchains are distributed systems that share a common state, the network must agree on the content of the distributed ledger. Therefore, a consensus model is needed in each

blockchain. It ensures that the next block in the chain is the one and only version of the truth. Using a consensus mechanism allows the blockchain to avoid a central authority that keeps track of all accounts. There are some key features consensus models must deliver to be usable in blockchain implementations:

Consistency A consensus mechanism is safe if all nodes produce the same valid output

Aliveness A consensus mechanism assures aliveness if all nodes participating eventually produce a result

Fault Tolerance A consensus mechanism delivers fault tolerance if it can recover from failure of a node

The figure 1.2[2] tells us how secure the information stored in bitcoin is.The consensus models makes the model even more secure.

[2]https://www.fin24.com/Finweek/Investment/should-you-buy-more-bitcoin-20180307-2

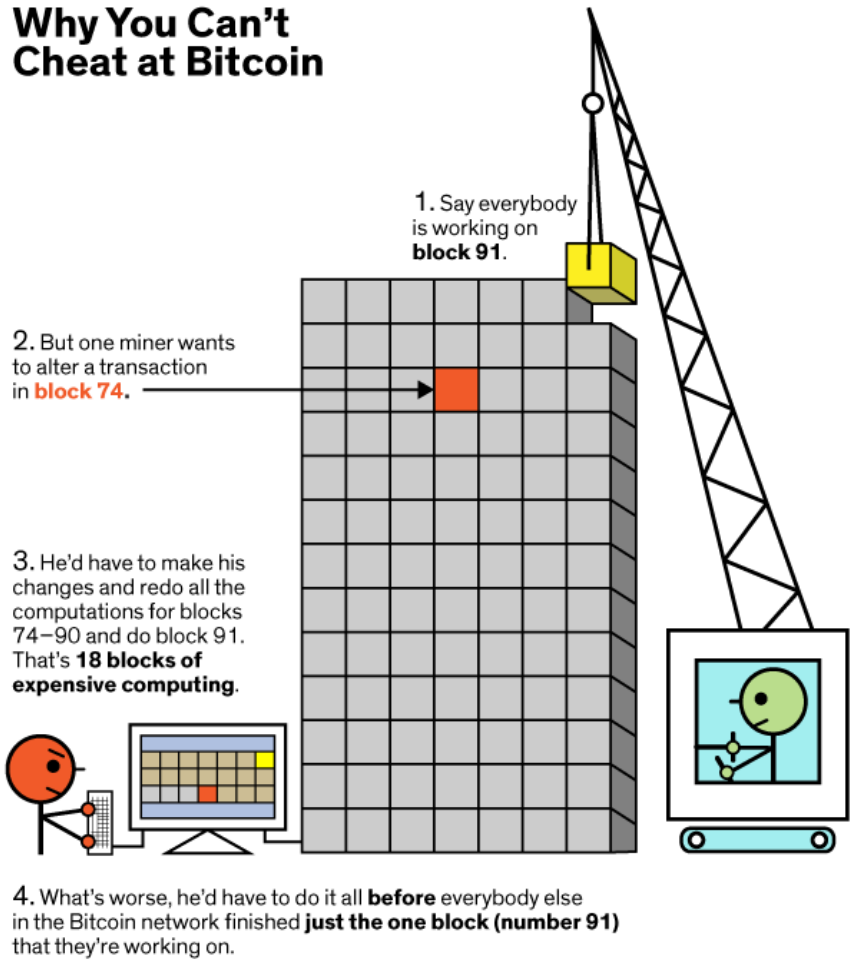

Figure 1.2: Consensus

1.4.1 Proof of Work

This is the most popular algorithm being used by currencies such as Bitcoin and Ethereum, each one with its own differences. The actor who will solve the aforementioned problem first the majority of the time is the one who has access to the most computing power. These actors are also called miners. It has been widely successful primarily due to its following properties:

- It is hard to find a solution for that given problem.
- When given a solution to that problem it is easy to verify that it is correct.
- Whenever a new block is mined, that miner gets rewarded with some currency (block reward, transaction fees) and thus are incentivized to keep mining.In Proof

of Work, other nodes verify the validity of the block by checking that the hash of the data of the block is less than a preset number.

- Due to the limited supply of computational power, miners are also incentivized not to cheat. Attacking the network would cost a lot because of the high cost of hardware, energy, and potential mining profits missed.

1.4.2 Proof of Stake

Proof of Stake takes away the energy and computational power requirement of PoW and replaces it with stake. Stake is referred to as an amount of currency that an actor is willing to lock up for a certain amount of time. In return, they get a chance proportional to their stake to be the next leader and select the next block. There are various existing coins which use pure PoS, such as Nxt and Blackcoin. The main issue with PoS is the so-called nothing-at-stake problem. Essentially, in the case of a fork, stakers are not disincentivized from staking in both chains, and the danger of double spending problems increase.

1.4.3 Proof of Authority

Proof of Authority (PoA) is an alternative consensus mechanism, which does not depend on the nodes solving arbitrarily difficult mathematical problems, but instead uses a set of ”authorities” - nodes that are explicitly allowed to create new blocks and secure the blockchain. The three main conditions that must be fulfilled for a validator to be established are:

- Identity must be formally verified on-chain, with a possibility to cross-check the information in a publicly available domain.

- Eligibility must be difficult to obtain, to make the right to validate the blocks earned and valued. (Example: potential validators are required to obtain public notary license)

- There must be complete uniformity in the checks and procedures for establishing an authority.

Chapter 2

Ethereum

2.1 Introduction

Ethereum [3] is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. It supports a modified version of Nakamoto consensus via transaction based state transitions.

In the Ethereum blockchain, instead of mining for bitcoin, miners work to earn Ether, a type of crypto token that fuels the network. Beyond a tradeable cryptocurrency, Ether is also used by application developers to pay for transaction fees and services on the Ethereum network.

2.2 Terminologies

2.2.1 Address

Ethereum addresses are composed of the prefix ”0x”, a common identifier for hexadecimal, concatenated with the rightmost 20 bytes of the Keccak-256 hash (big endian) of the ECDSA public key. In hexadecimal, 2 digits represents a byte, meaning addresses contain 40 hexadecimal digits. For example, the Poloniex ColdWallet $0xb794F5eA0ba39494cE839613fffBA74279579268$.

2.2.2 Ethereum VM

The Ethereum Virtual Machine focuses on providing security and executing untrusted code by computers all over the world. To be more specific, this project focuses on preventing Denial-of-service attacks, which have become somewhat common in the

cryptocurrency world. Moreover, the EVM ensures programs do not have access to each other's state, ensuring communication can be established without any potential interference.

To put this into a language everyone can understand, the Ethereum Virtual Machine is designed to serve as a runtime environment for smart contracts based on Ethereum. As most cryptocurrency enthusiasts are well aware of, smart contracts are very popular these days. This technology can be used to automatically conduct transactions or perform specific actions on the Ethereum blockchain. Many people predict smart contracts will help revolutionize finance and other industries over the coming years.

2.2.3 Gas

"Gas" is the name for a special unit used in Ethereum. It measures how much "work" an action or set of actions takes to perform: for example, to calculate one Keccak256 cryptographic hash it will take 30 gas each time a hash is calculated, plus a cost of 6 more gas for every 256 bits of data being hashed. Every operation that can be performed by a transaction or contract on the Ethereum platform costs a certain number of gas, with operations that require more computational resources costing more gas than operations that require few computational resources.

The reason gas is important is that it helps to ensure an appropriate fee is being paid by transactions submitted to the network. By requiring that a transaction pay for each operation it performs (or causes a contract to perform), we ensure that network doesn't become bogged down with performing a lot of intensive work that isn't valuable to anyone. This is a different strategy than the Bitcoin transaction fee, which is based only on the size in kilobytes of a transaction. Since Ethereum allows arbitrarily complex computer code to be run, a short length of code can actually result in a lot of computational work being done. So it's important to measure the work done directly instead of just choosing a fee based on the length of a transaction or contract.

So if gas is basically a transaction fee, how do you pay it? This is where it gets a little tricky.Although gas is a unit that things can be measured in, there isn't any actual

token for gas. That is, you can't own 1000 gas. Instead, gas exists only inside of the Ethereum virtual machine as a count of how much work is being performed. When it comes to actually paying for the gas, the transaction fee is charged as a certain number of ether, the built-in token on the Ethereum network and the token with which miners are rewarded for producing blocks.

2.2.4 Ether

Ether is a fundamental cryptocurrency for operation of Ethereum, which thereby provides a public distributed ledger for transactions. It is used to pay for gas, a unit of computation used in transactions and other state transitions.

2.3 Ethereum vs Bitcoin

Ether is different from Bitcoin in several aspects:

- Its block time is 14 to 15 seconds, compared with 10 minutes for bitcoin.
- Mining of ether generates new coins at a usually consistent rate, occasionally changing during hard forks, while for bitcoin the rate halves every 4 years.
- For proof-of-work, it uses the Ethash algorithm which reduces the advantage of specialized ASICs in mining.
- Transaction fees differ by computational complexity, bandwidth use and storage needs (in a system known as gas), while bitcoin transactions compete by means of transaction size, in bytes.
- Ethereum gas units each have a price that can be specified in a transaction. This is typically measured in Gwei. Bitcoin transactions usually have fees specified in satoshis per byte.
- Transaction fees are generally considerably lower for ether than for Bitcoin. In December 2017, the median transaction fee for ether corresponded to $0.33, while for bitcoin it corresponded to $23.

- Ethereum uses an account system where values in Wei are debited from accounts and credited to another, as opposed to Bitcoin's UTXO system, which is more analogous to spending cash and receiving change in return. Both systems have their pros and cons; in terms of storage space, complexity, and security/anonymity.

2.4 Testnets

Testnets are copies of the Ethereum blockchain almost identical in every way to the Mainnet except in the fact that their Ether is worthless (and, of course, the software that's been deployed on these testnets). There are three types of testnets which are explained as follows.

2.4.1 Public Test

Public testnets are available to everyone, they're connected to the internet. Anyone can connect to them at any time, even from popular wallet interfaces like MyEtherWallet or MetaMask.

Rinkeby:

Launched in April 2017 by the Ethereum team, Rinkeby shares the advantages of Kovan with two minor alterations: it does not support Parity and only works with Geth, and it uses a slightly different PoA consensus mechanism.Rinkeby is also supported on Etherscan: $https://rinkeby.etherscan.io/$ Rinkeby Ether can be requested from an authorized faucet.

Ropsten:

Ropsten [4] was launched in November 2016. Its Ether can be mined just like on the Mainnet. Both Geth and Parity support it – two different implementations of the Ethereum node software – so it's possible to develop for it from two different angles.Of all three testnets, Ropsten resembles the current Mainnet the most. Its results resemble Mainnet results because its consensus mechanism is PoW (i.e. it can be mined on) so the simulation of transaction confirmations is the most realistic.

Ropsten's current blockchain file size is around 9 GB.On the Ropsten network, Ether can be mined or demanded through the Ropsten Faucet– a website that exists for the sole purpose of giving away free test Ether. Because Ether can be mined on Ropsten, it is susceptible to spam attacks – a wave of useless transactions that clog the network. If Ether is free and easy to get, it's easy to create this flood. Such an attack happened in February 2017, when attackers mined enormous amounts of Ether and kept sending too big transactions into the network. Ethereum's block size limit is designed to be flexible and to grow with demand, so they managed to pump it up to several billion units of gas instead of the 4 million it was at up until then. After this pumping of block size limits, they sent large computationally intensive but useless transactions into the network, clogging the gas limit in those immense blocks completely, inflating blockchain size for no reason and blocking everyone else's work. The network collapsed and was revived a month later after resetting the blockchain data.

It's interesting to note that Ropsten only differs from the Mainnet (on which all of us hold our "real" Ether) by agreement. We collectively decided that Ropsten's Ether is worthless, and now it is. Ropsten has its own mining pools, its own software, etc. but we all decided that its Ether has no value, and so it doesn't. Here we see, quite literally, how the community's consensus dictates the value of an asset – or, to be more exact, the lack of value of said asset.

2.4.2 Private Test

A private test network is equivalent to one's own personal blockchain – your own copy of Ethereum. When booting up a private blockchain, a Genesis file needs to be generated from which a tool like Geth builds the new chain. This chain is then inspected and interacted with via tools like Mist, MetaMask, MyEtherWallet, etc.

Figure 2.1: Private Testnet

Private testnets are excellent for teamwork and closed environments that need to simulate mining and transaction confirmations without exposing their network to the outside world and risking spam attacks. There's no expense involved in creating one, other than a small fraction of the CPU and disk space of the developer's computer being occupied while the testnet is in use. After a private testnet grows enough, it can be exposed to the public through the internet and other interested parties can connect to it and extend it. This lends itself perfectly to experimentation, collaboration, cross-application interaction, and more. Ether can be mined on even the weakest computers when running a private testnet, and during initialization some addresses can even get some Ether pre-mined for future use if needed.

2.4.3 TestRPC

Testrpc is a NodeJS package which simulates the Ethereum network on a single computer. When launching, it'll generate several Ethereum addresses, each with some Ether already on it. Testrpc – though a cool idea in theory – usually falls flat in practice due to some bugs and makes for a particularly frustrating experience. In our experience, it's easier and more reliable to just launch a private testnet as shown in figure 2.1 with all the bells and whistles of a proper blockchain, than to bother with setting up Testrpc.

2.5 Smart Contracts

Smart contracts help you exchange money, property, shares, or anything of value in a transparent, conflict-free way while avoiding the services of a middleman. The best

way to describe smart contracts is to compare the technology to a vending machine. Ordinarily, you would go to a lawyer or a notary, pay them, and wait while you get the document. With smart contracts, you simply drop a bitcoin into the vending machine (i.e. ledger), and your escrow, driver's license, or whatever drops into your account. More so, smart contracts (Figure 2.2[1]) not only define the rules and penalties around an agreement in the same way that a traditional contract does, but also automatically enforce those obligations.

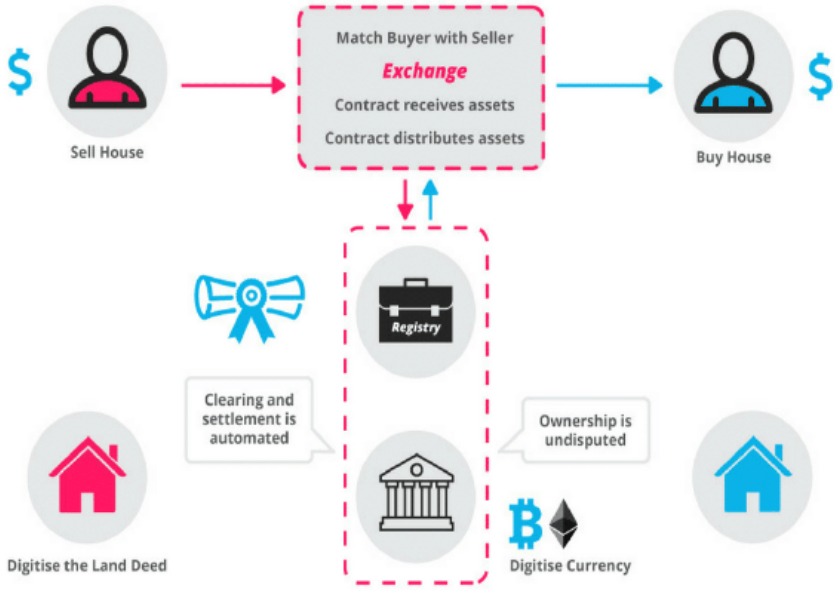

Figure 2.2: Working of Smart Contract

For Example, Suppose you rent an apartment from me. You can do this through the blockchain by paying in cryptocurrency. You get a receipt which is held in our virtual contract; I give you the digital entry key which comes to you by a specified date. If the key doesn't come on time, the blockchain releases a refund. If I send the key before the rental date, the function holds it releasing both the fee and key to you and me respectively when the date arrives. The system works on the If-Then premise and is witnessed by hundreds of people, so you can expect a faultless delivery. If I give you the key, I'm sure to be paid. If you send a certain amount in bitcoins, you receive the key. The document is automatically canceled after the time, and the code

[1]https://blockgeeks.com/guides/smart-contracts/

cannot be interfered by either of us without the other knowing since all participants are simultaneously alerted.

You can use smart contracts for all sort of situations that range from financial derivatives to insurance premiums, breach contracts, property law, credit enforcement, financial services, legal processes and crowdfunding agreements.

2.6 Ethereum Wallet

A cryptocurrency wallet is a software program that stores private and public keys and interacts with various blockchain to enable users to send and receive digital currency and monitor their balance. If you want to use Bitcoin or any other cryptocurrency, you will need to have a digital wallet.

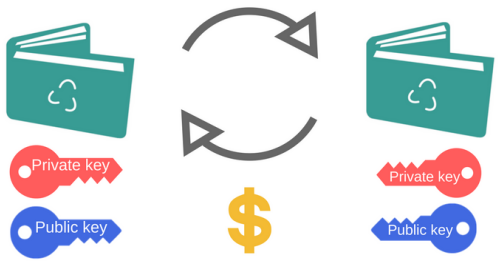

Figure 2.3: Wallet

The Ethereum Wallet (Figure 2.3[2]) is a gateway to decentralized applications on the Ethereum blockchain. It allows you to hold and secure ether and other crypto-assets built on Ethereum, as well as write, deploy and use smart contracts. Millions of people use cryptocurrency wallets, but there is considerable misunderstanding about how they work. Unlike traditional 'pocket' wallets, digital wallets don't store currency. In fact, currencies don't get stored in any single location or exist anywhere in any physical form. All that exists are records of transactions stored on the blockchain.

[2]https://blockgeeks.com/guides/cryptocurrency-wallet-guide/

Chapter 3

Ethereum DApp

3.1 Web 2.0 vs Web 3.0

Areas	Web 2.0	Web 3.0(DAPPS)	Status
Scalable computation	Amazon EC2	Ethereum,Truebit	Inprogress
File storage	Amazon S3	IPFS/Filecoin,storj	Inprogress
External Data	3rd Party APIs	Oracles(Augur)	Inprogress
Monetization	Ads,selling goods	Token model	Ready
Payments	Credit cards,Paypal	Ethereum,Bitcoin	Ready

Table 3.1: Web 2.0 vs Web 3.0

The table 3.1 gives a list of tools used in centralized networks(Web 2.0),their counterparts in Decentralized networks(Web 3.0) and their current status of transition.

3.1.1 Architecture of Ethereum

The Architecture of ethereum is shown in figure 3.1[1]. Consensus layer refers to a protocol which describes the ledger format and a consensus function that can be used to determine which of the multiple candidate ledgers is the consensus ledger. Economic layer helps to create tokens that incentivise the nodes to perform computation and other specific functionalities. Blockchain services contains the programs or codes which do the process. Interoperability layer contains exchange protocols that takes care of exchanging information between nodes in the network and making use of exchanged messages. Browsers are used to access the decentralized applications.

[1]https://read.acloud.guru/the-blockchain-stack-get-ready-for-big-things-to-come-545d6270b6ab

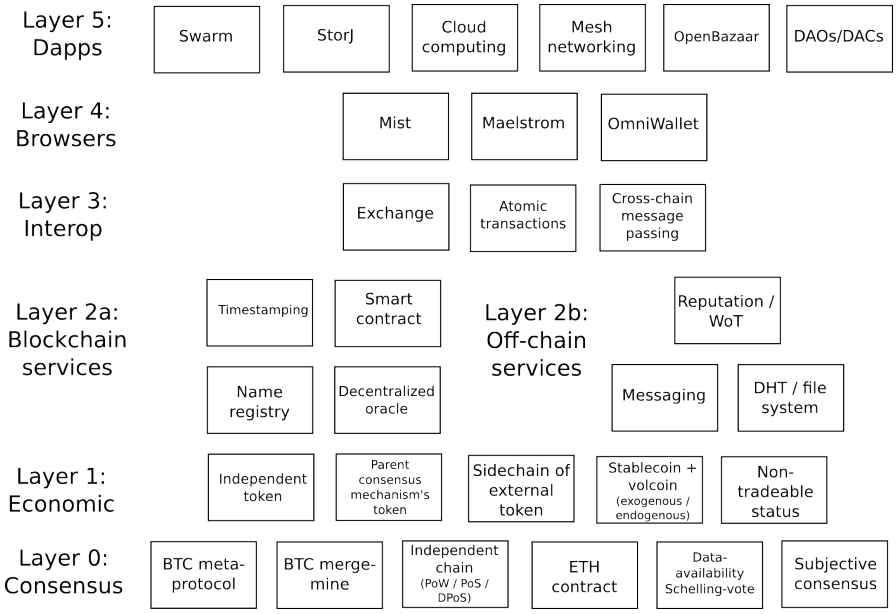

Figure 3.1: Architecture of Ethereum

3.2 Ethereum Client

Ethereum network is composed of ethereum nodes that are connected to each other.Ethereum nodes send or receive blocks of data to their peers and they perform validation and mining process.Decentralized applications connect to ethereum nodes for sending transactions.Transactions may be used for transferring ethers or for deploying or executing contracts. The reference implementation for ethereum client is available in many languages like c++(eth),python(pyethapp) and golang(geth). There are also some third party implementations available.

3.2.1 Geth

Geth is the implementation of Ethereum client (Fig : 3.2[2]) using Golang. It is a command line tool. There are various commands in geth which are used for managing and running a full ethereum node. The general syntax of geth usage is: Geth [options] command [command options] [args]

[2]https://decentralize.today/ethereum-on-a-iphone-9ca7b41a28bd

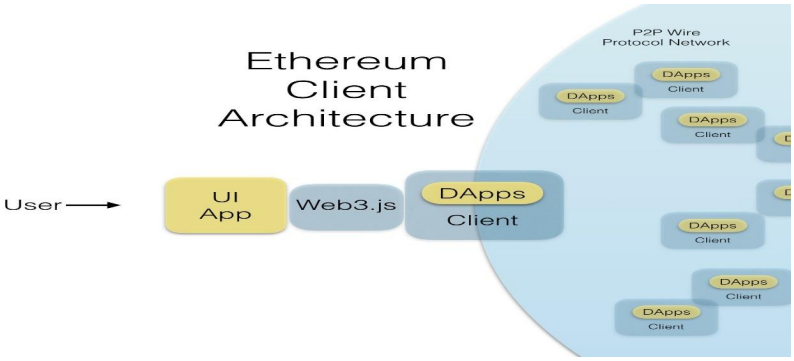

Figure 3.2: Ethereum Client Architecture

3.3 Web3 Javascript API

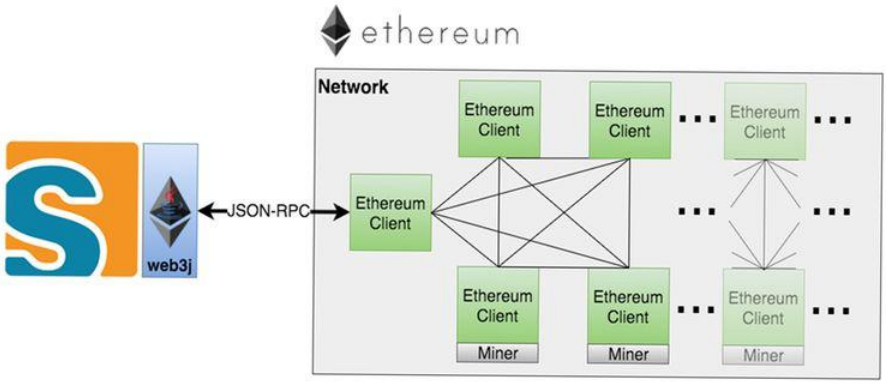

Figure 3.3: Web3J

Decentralized applications [5] need to connect to Ethereum nodes.Multiple libraries are available to simplify the process of connecting to ethereum nodes.Web3 JS as shown in figure 3.3[3] [6] is the javascript library. Ethereum clients support javascript run time environment and web3 api.Geth in addition to above apis supports apis like management api also which are used for managing and controlling nodes. Web3 apis are used for building decentralized applications on ethereum.They can be both interactive and non interactive.In web3 api and management api, methods are grouped under objects based on functions they provide.Shorthands are also available for apis.

[3]https://web3j.readthedocs.io/en/latest/

Eth	Ethereum blockchain related methods
Net	Nodes network status
Db	Read/write in local db
Version	Version information of node connected

Table 3.2: Web3

Admin	Node Management
Personal	Account management
Miner	Miner Management

Table 3.3: Management

Web3 javascript apis act as interface between front end of the decentralized application and its backend which contains the smart contracts as shown in figure 2 above.Compilation using Web3 Api is supported only till geth version 1.5.9. Web3.eth.compile.solidity(string,callback_ func) is the function for compiling contracts.

3.4 Frameworks

3.4.1 Remix

Remix is the browser based development environment for ethereum contracts as shown in figure 3.It has an integrated compiler and solidity runtime environment.It has simple user interface for writing and managing contracts.The compilation and deployment process are not automated here.Remix is suitable for simple contracts. Javascript virtual machine(JVM),connecting to a web3 provider and an injected web3 instance(like metamask) are three ways in remix with which you can inject a blockchain into browser.

3.4.2 Ethereum Simulators

Ethereum simulators are open source tools which are used to test the smart contracts as a part of development process.They create networks identical to ethereum network and are not connected to real time ethereum node.All the executions takes place in the sandbox of simulator.No mining is involved here.Therefore it is a faster way for deploying and testing smart contracts during development phase.

3.4.2.1 Ganache

Ganache [7] is an ethereum simulator and a personal blockchain engine which is a part of truffle framework.Ganache is available as both command line interface and graphical user interface.Ganache CLI is replacement for testRPC.Ganache CLI are used for automation and tooling.The working,debugging and setting up configurations are easier in Ganache GUI. It has built in block explorer which is used to see information about the new blocks created and the transactions executed.

3.4.3 Mocha

Mocha is a javascript testing framework.It runs on Nodes js platform.Any assertion library is supported by mocha.Browser support,test coverage reports,asynchronous testing are features of mocha.

3.4.4 Chai

Chai is an open source test driven development/behaviour driven development assertion library for browser and node.It can be paired with any testing framework.Users are given choices to choose for interfaces of chai library.TDD style provides classical feel while BDD styles provide readable style.

3.4.5 Metamask

Metamask [8] is a browser extension which allows to run ethereum decentralized applications in your browser without running a full ethereum node.It injects web3 api into javascript context of every webpage, so that decentralized applications can be read from the blockchain. It also lets the user create and manage their own identities, so when a Dapp wants to perform a transaction and write to the blockchain, the user gets a secure interface to review the transaction, before approving or rejecting it.

3.5 Solidity

Smart contracts can be written in many languages.It can be written in serpen or list or solidity.The popular language for contract writing is Solidity [9].

3.5.1 Variables

Solidity is a static language so the types of variables are declared in compile time itself.Booleans,integers,Address,Fixed size arrays are some of the frequently used variable types.Complex types can be formed by combining several elementary types.

```
address x = 0x123;
uint128 a = 1;
```

3.5.2 Functions

Solidity has two types of function namely internal and external.Functions which can be called only inside a current contract are internal functions.Default functions are internal functions.

```
function (<parameter types>) {internal|external} [pure|constant|view|
    payable] [returns (<return types>)]
```

3.5.3 Arrays

Solidity supports storage arrays and memory arrays.Solidity arrays can have fixed size or dynamic size.Multidimensional arrays can also be created.Special arrays are of bytes and string data type.The element type is arbitrary for storage arrays and is not mapping for memory arrays.

```
contract Sample {
  uint[] memory a = new uint[](7);
         bytes memory b = new bytes(len);
}
```

3.5.4 Structs

New types in the form of structs can be defined in solidity.Structs can contain all the supported data type variables along with arrays and mappings but it cannot contain a member of its own type.

```
contract CrowdFunding {
    struct Funder {
        address addr;
        uint amount;
    }
}
```

3.5.5 Mappings

Mappings are declared as mapping(keytype => valuetype).Valuetype can be any supported data type including mapping and Keytype can be of any type except a contract,a struct,an enum,a dynamically sized array and a mapping.Mappings can be viewed as hash table where each key is mapped to a value.

```
contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
```

3.5.6 Contract Deployment

There are two parts in deploying smart contracts in ethereum network namely,

- Bytecode
- Abi definition

The binary that gets deployed in the network is bytecode. Application binary interface is a json format file which contains information about public interfaces of the ethereum contract like events emitted and functions exposed by the contract.Abi definition are needed both for deployment and invocation of contracts.

Chapter 4

Banking Decentralized Application

4.1 Objective

The main objectives for developing a decentralized banking application are:

- To ensure immutability of bank details once authorised
- To implement transparency of bank processing to certain allowable extents
- To reveal the authority in charge of the respective banking process to the public by storing the pubic key and the digital signature of the authority alongside the information
- To ultimately prevent fraudulent activites in the banking sectors caused by systemic failures in detecting human malfeasance

4.2 Key Features

The Banking DApp mainly focuses on decentralising the loan and tender process. All the participants (users, bank officials) are assumed to have registered with a Certification Authority and been assigned a RSA key pair { public key, private key } which is unique to them. At the user end,

i The user $(Key : [U_{pub}, U_{pri}])$ creates an account in the Banking website by entering his/her Public Key U_{pub} along with the basic information for KYC.

ii Then, the Loanee / Representative of an organisation has to digitally sign a fixed message (previously set by the bank and updated periodically) using his/her Private key U_{pri} before applying for loan or tender.

$$Sig_U = msg^{U_{pri}}(modN)$$

This RSA signature is stored in the user database for the bank officials to authenticate the identity of user while processing loan or tender. Once registered as mentioned above, the user or organisation visits the banking sector with the necessary details. At the administration end,

The bank officials verify the user signed signature using the corresponding public key of user.

$$Sig_U^{U_{pub}} \equiv msg(modN)$$

If the verification is a success, then

- The bank officials $(Key : [B_{pub}, B_{pri}])$ digitally sign the details of loan or tender using their private key and upload the signed details in the blockchain. In case of loan, the block hash is stored in the user database.
- When the required number of bank officials authorise a particular loan (i.e) more block hashes are stored for a particular loan, the loan is sanctioned.
- In case of tender, the best quotation quoted is selected as the winner.

If found to be invalid, the processing truncates immediately. Once the details are pushed in the blockchain, the details become immutable and transparent. The authority signature stored alongside the details ensures non-repudiation by the authority in the future. Moreover, multiple authorised block confirmations for a loan is required for it to be sanctioned. Hence, multiple authentications and verifications are done before sanctioning loans thus augmenting more security by decentralising the banking process.

4.3 Implementation Approach

We have followed the approach of developing, implementing and testing the DApp in the ethereum simulator Ganache and testnet Ropsten. Both the approaches are explained with demo pictures of the DApp UI in this section. The back end and front end codes are explained in the Appendix.

4.3.1 Ganache-CLI

Primarily, we launch the Ganache Ethereum Simulator using ganache-cli command in the command prompt as shown in the figure 4.1.The simulator then creates 10 accounts which resemble the nodes in the blockchain.

Figure 4.1: Starting Ganache

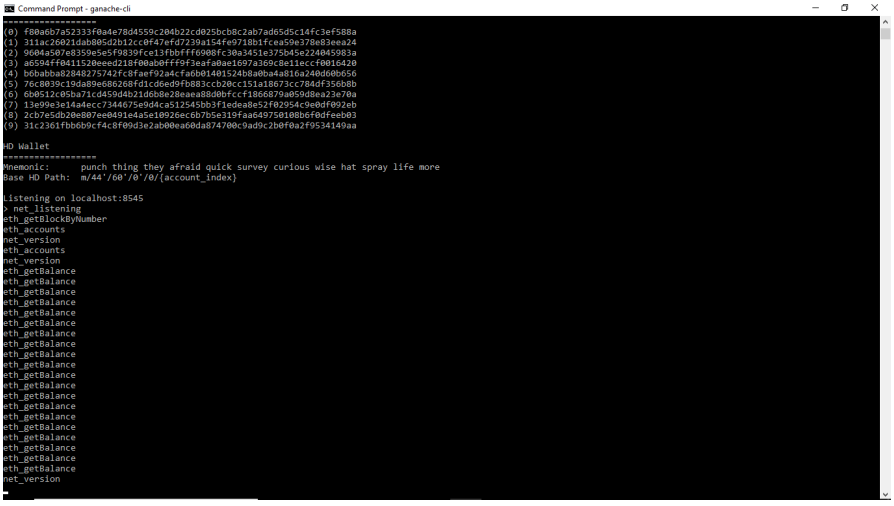

Figure 4.2: Connecting Ganache with Remix

As the next step, connect to the Remix IDE by selecting the option Web3 Provider localhost:8545 as shown in figure 4.2. Then start the Apache HTTP server and MySql using XAMPP and handle the administration using phpMyAdmin as shown in figure 4.3.

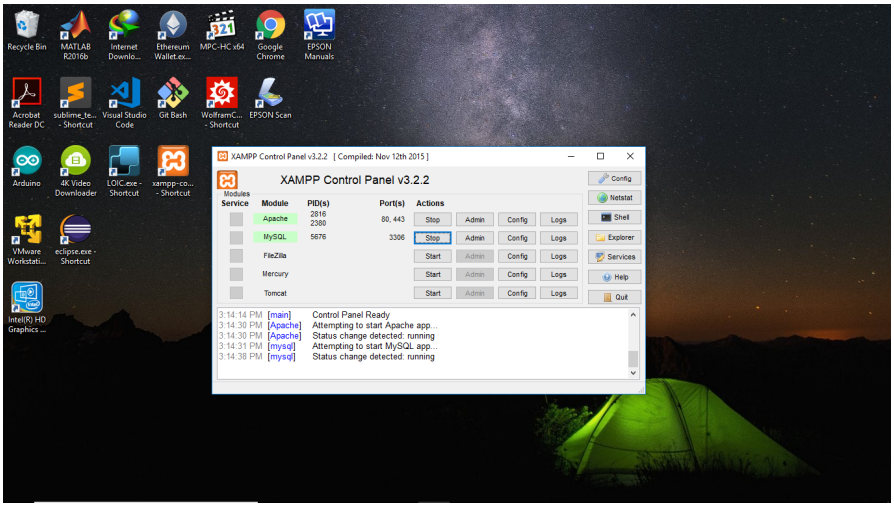

Figure 4.3: Bank Database using XAMPP Phpmyadmin

The first UI of our DApp is the sign up / login page as shown in figure 4.4. The phpMyAdmin UI is shown in figure 4.5 after the user signing up.

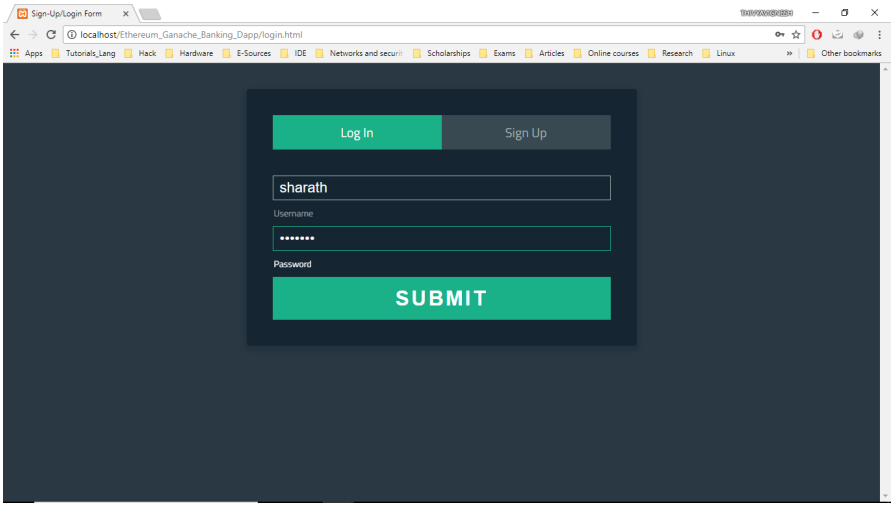

Figure 4.4: Sign Up & Login Page

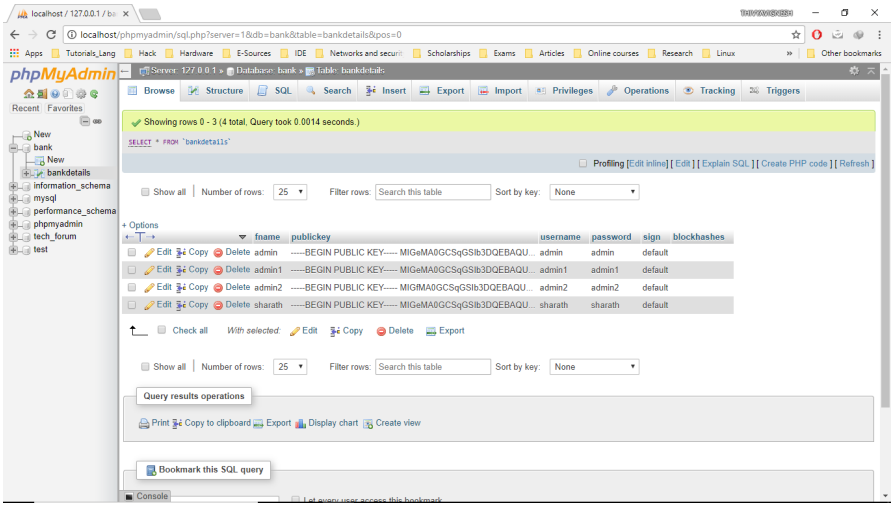

Figure 4.5: Bank Central Database after Sign Up by user

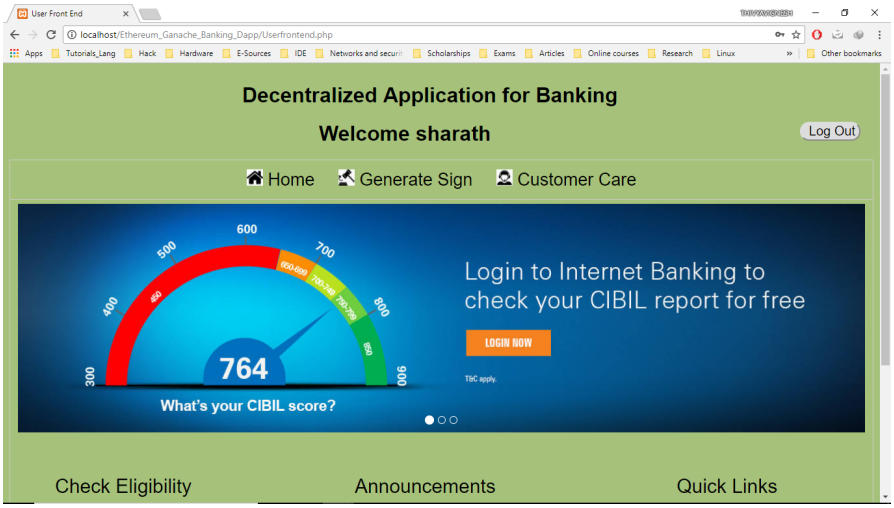

Figure 4.6: User Welcome Page

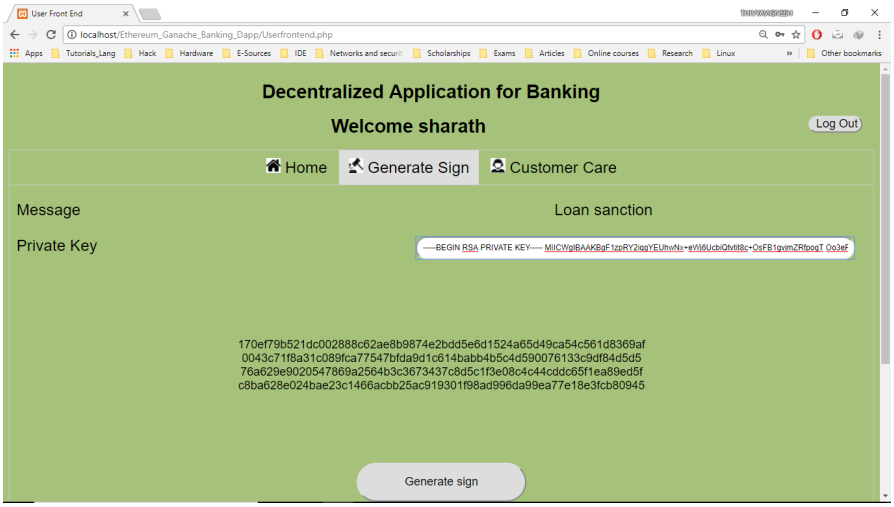

Figure 4.7: Message Signing by user

Once the user logs in, the welcome page appears as in figure 4.6. Then he/she has to sign a message using their private key as shown in figure 4.7 and this signature is then stored in the bank database as shown in figure 4.8.

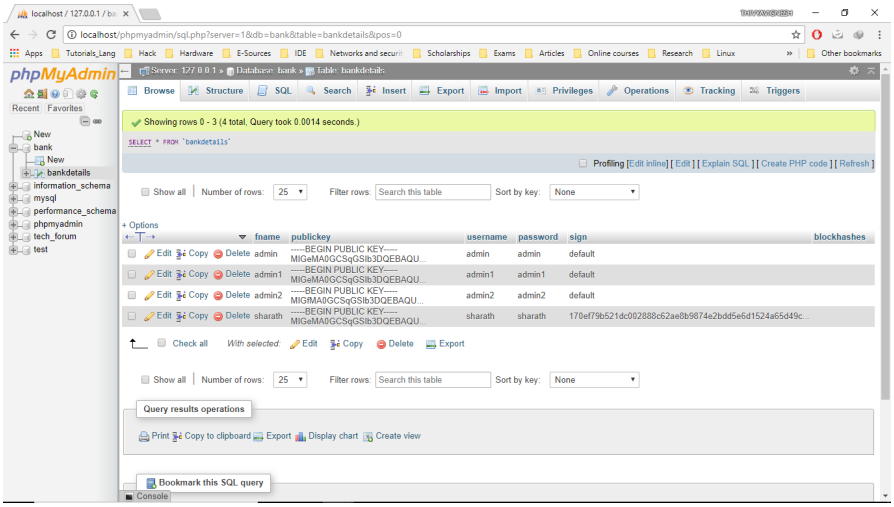

Figure 4.8: Bank Database after updation of user's signature

Then the bank official who is the authenticator logs in and enters the username of the user whose loan/tender details he would authenticate as shown in figure 4.9. As a result, the public key and the signature of the user saved are displayed on the screen for authentication of user by the bank official as shown in figure 4.10. If successful authentication goes to the admin page Loan/Tender else shows invalid (Figs 4.11 & 4.12).

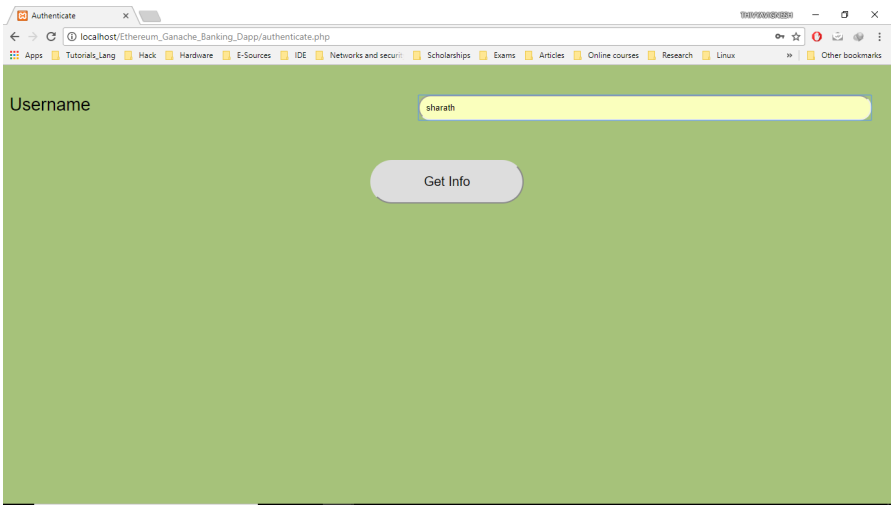

Figure 4.9: Authenticator Welcome Page

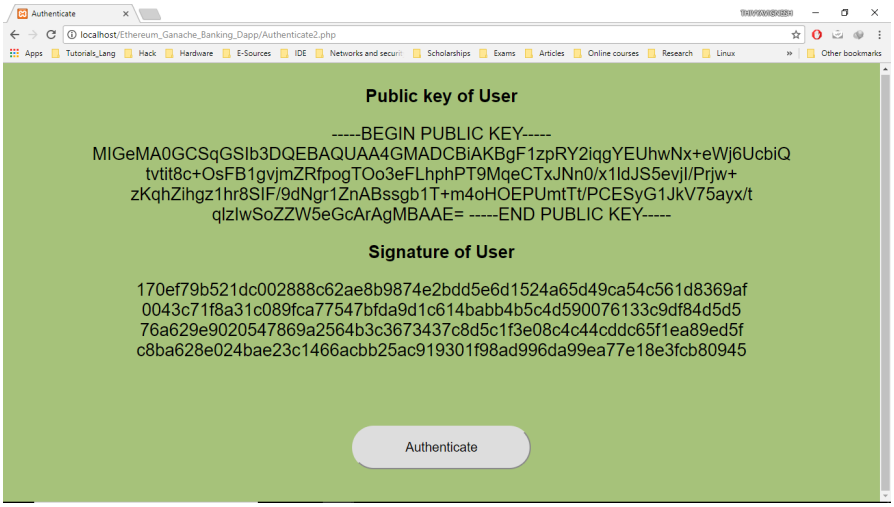

Figure 4.10: User Details page

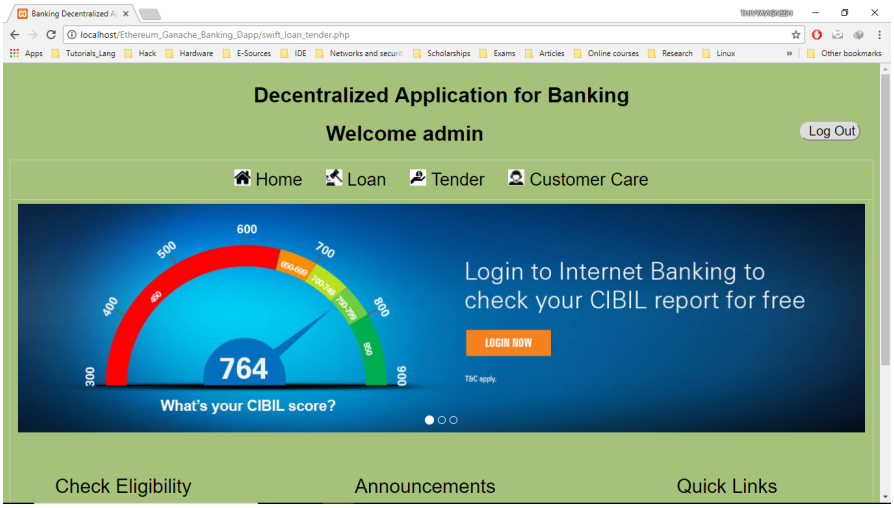

Figure 4.11: Authenticator Loan/Tender Page

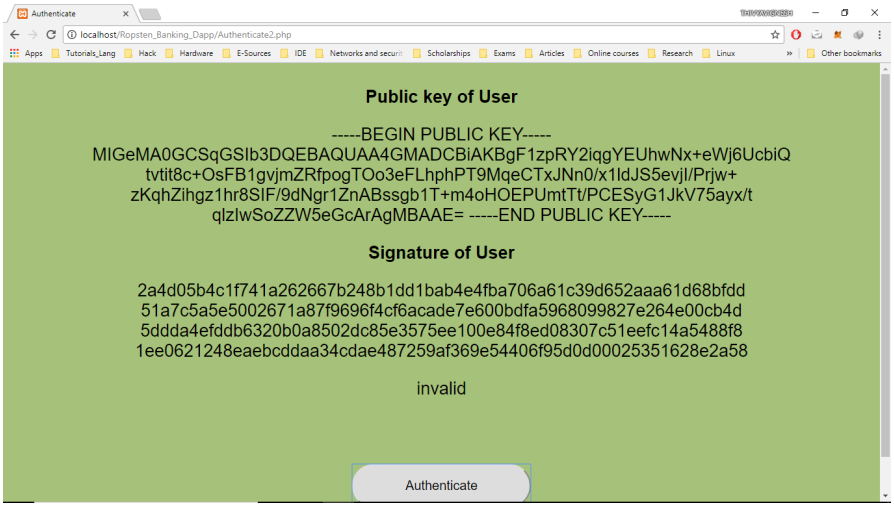

Figure 4.12: Invalid Authentication

The figure 4.13 shows the authenticator entering the loan details and signing the details by clicking the button Generate sign. When the Update Loanee button is clicked the block hash along with the details of the loan is displayed as shown in the figure 4.14. The block hash is then stored in the bank database (Fig 4.15). Figure 4.16 shows the display of loan details corresponding to the entered Loan ID.

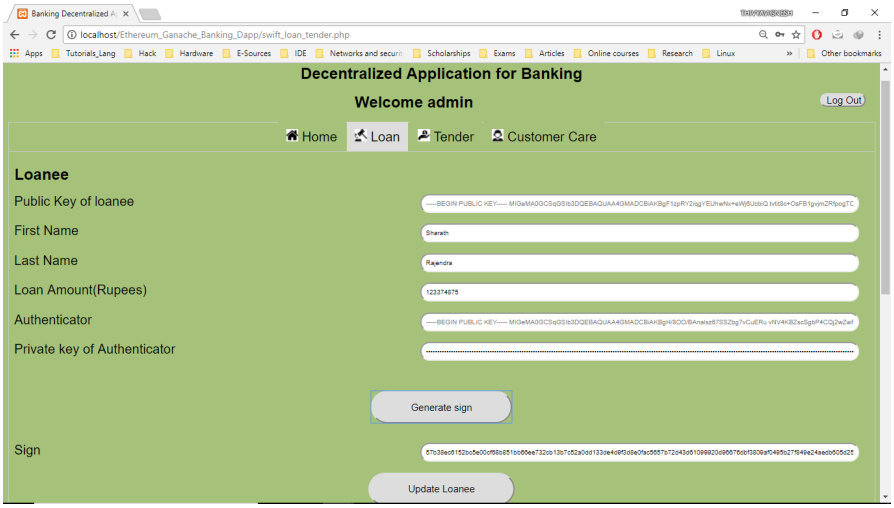

Figure 4.13: Authenticator enters Loan details

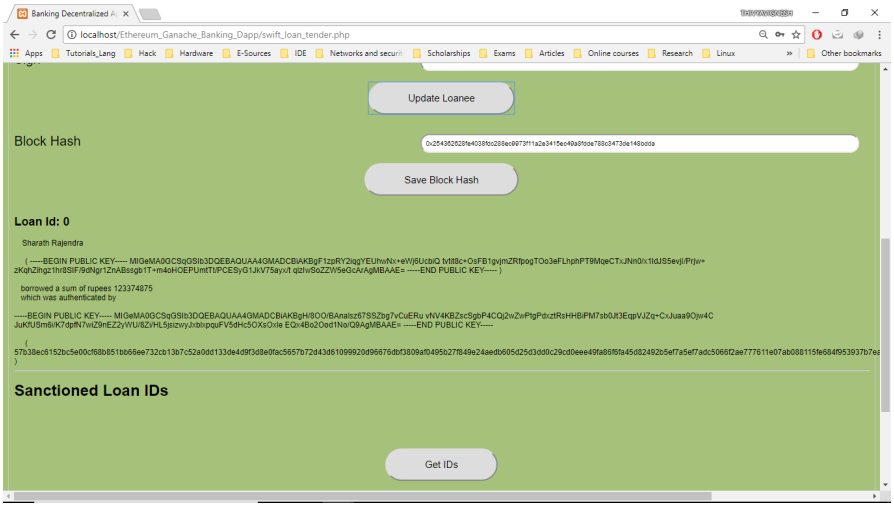

Figure 4.14: Save Block Hash of Loan Details

Similar to the entry of loan details and adding it to the blockchain, the tender details are also enterd and added to the blockchain as shown in the figure 4.17.

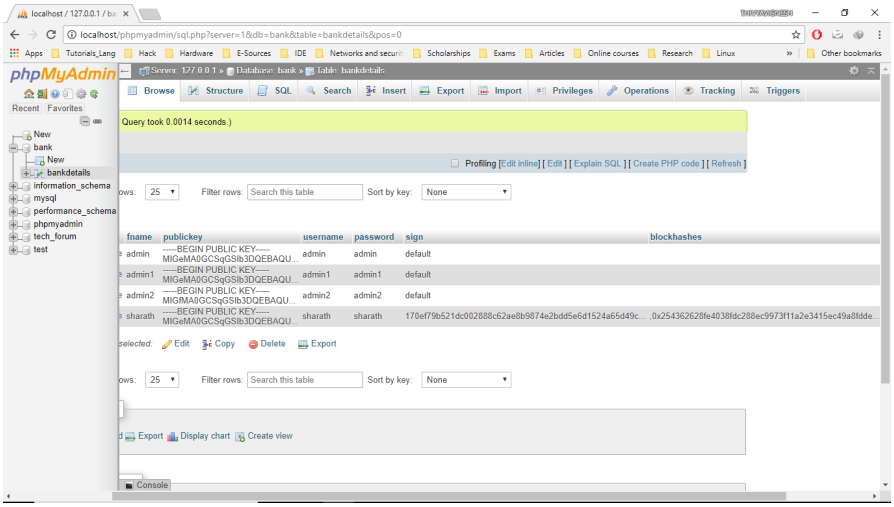

Figure 4.15: Bank Database after updation of BlockHash

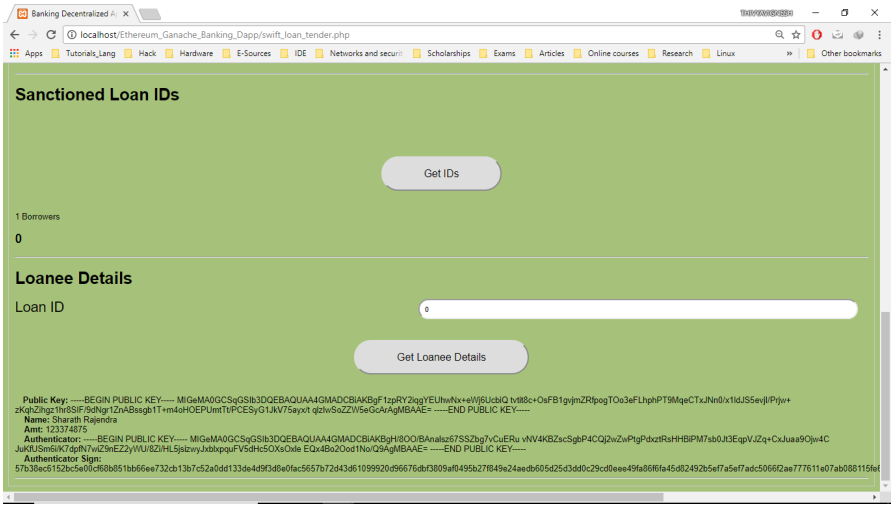

Figure 4.16: Loan details corresponding to Loan Id

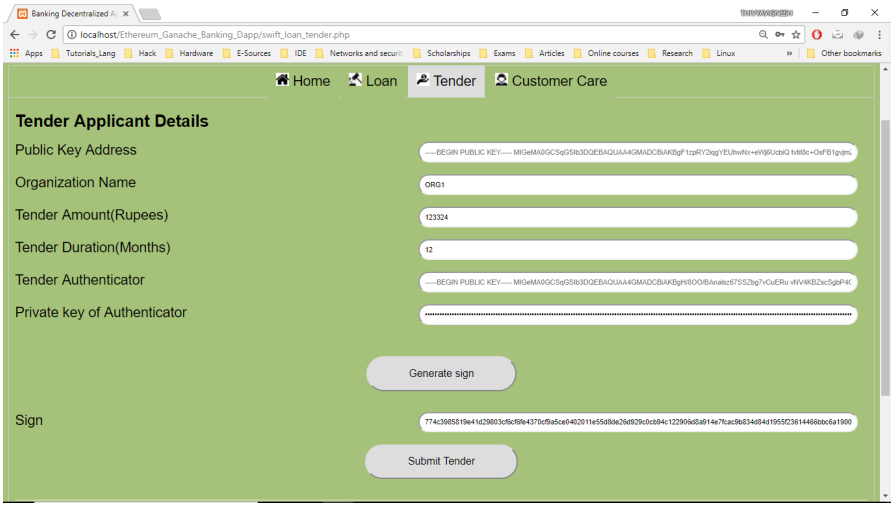

Figure 4.17: Authenticator enters Tender details

When clicked on the Get Tender Winner Button, if the maximum tender count is reached then the optimum tender quoted is displayed as shown in the figure 4.18 else NULL is displayed. The solidity contract here is compiled using Remix IDE (Fig 4.19).

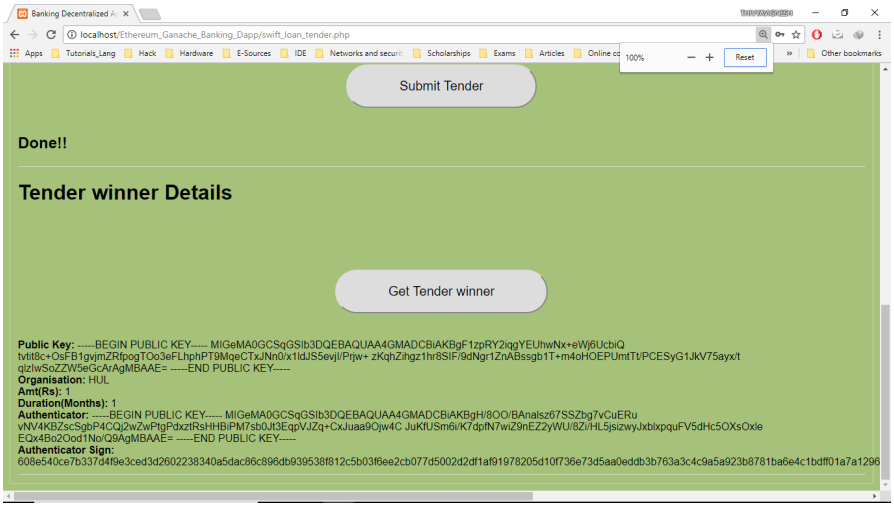

Figure 4.18: Displaying Tender Winner

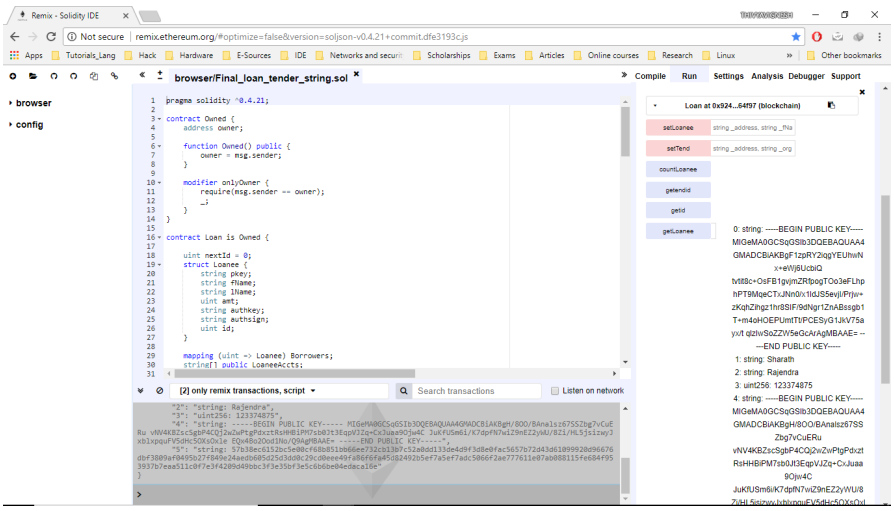

Figure 4.19: The smart contract using Remix IDE

For sanctioning of loan, the bank official looks into the user's record in the bank database. If more than the desired threshold count of blocks are confirmed by multiple authenticators which is equivalent to having multiple block hashes saved (Fig 4.20), then the loan is sanctioned. By doing so multi verification is done.

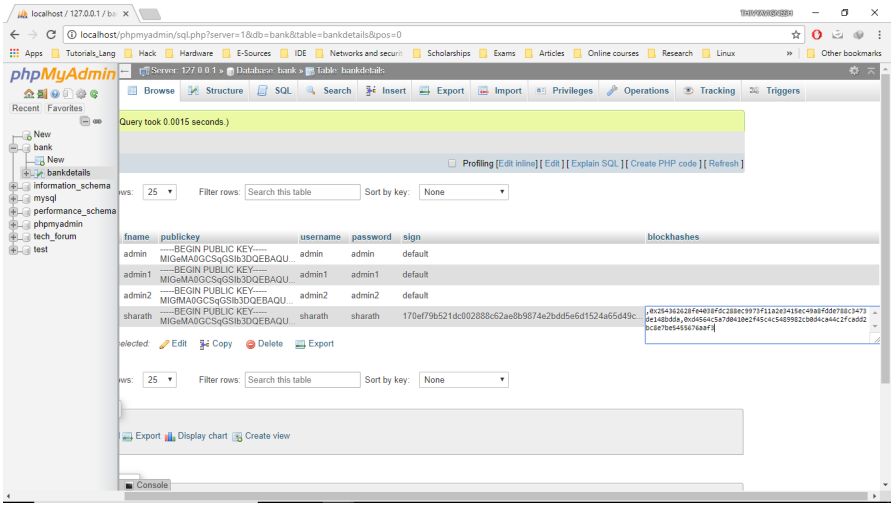

Figure 4.20: Multiple Verifications using multi block hash authentications

4.3.2 Ropsten Testnet

In the second approach, Ropsten testnet is used. The UIs are same as shown in the Ganache-CLI approach. Here the metamask chrome extension is integrated with the remix IDE by selecting the option of injected Web3 instance shown in the figure 4.21. The metamask extension simulates the Gas utilisation and expects fee payment through ropsten ethers(no value in real life) as implemented in the live net when a transaction is called(Fig 4.22).

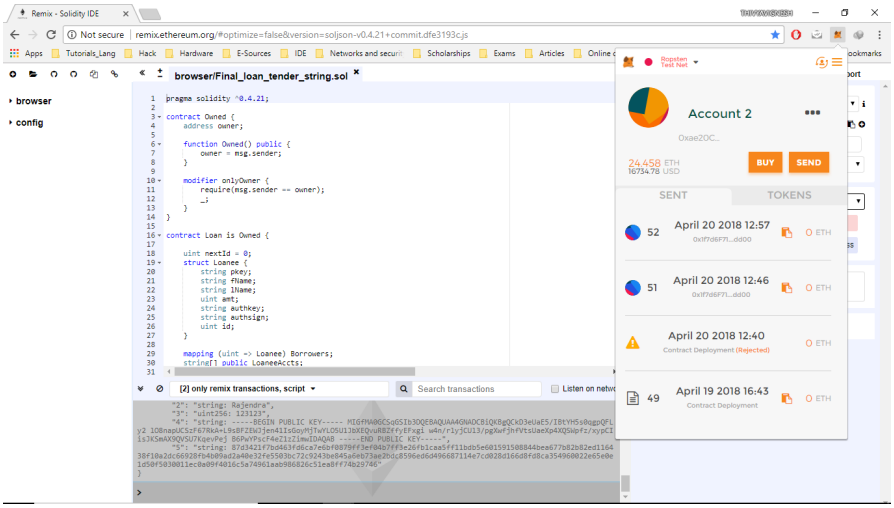

Figure 4.21: Metamask and Remix Integrated

The figures 4.23 and 4.24 captures the scenario when the setLoanee transaction is under processing and when set successfully.

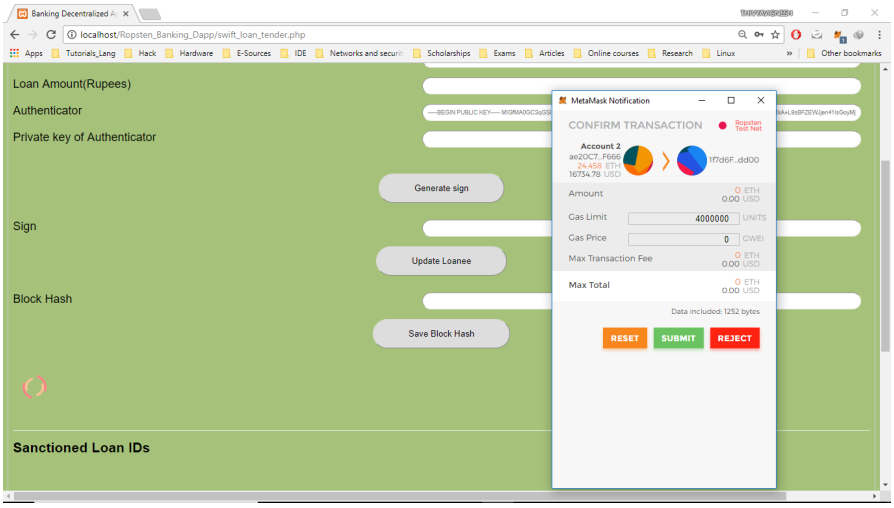

Figure 4.22: On updating the Loanee details to Ropsten by Authenticator

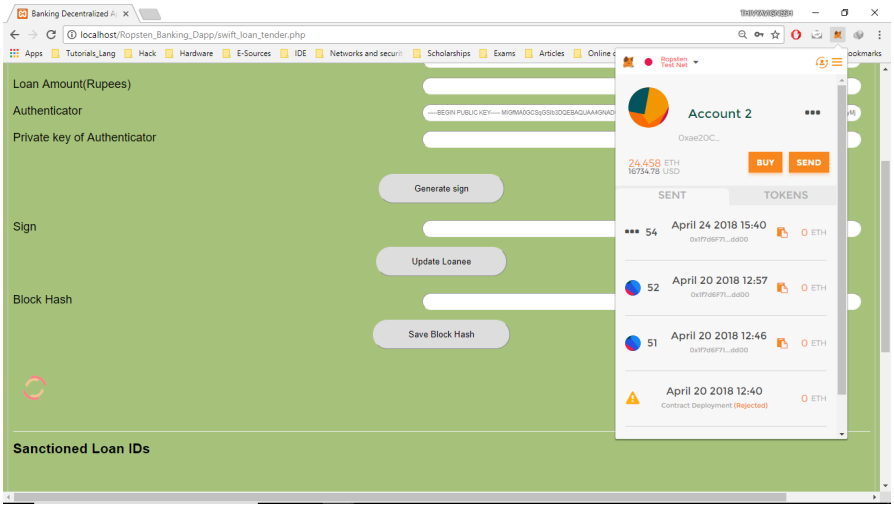

Figure 4.23: Processing the Transaction Request

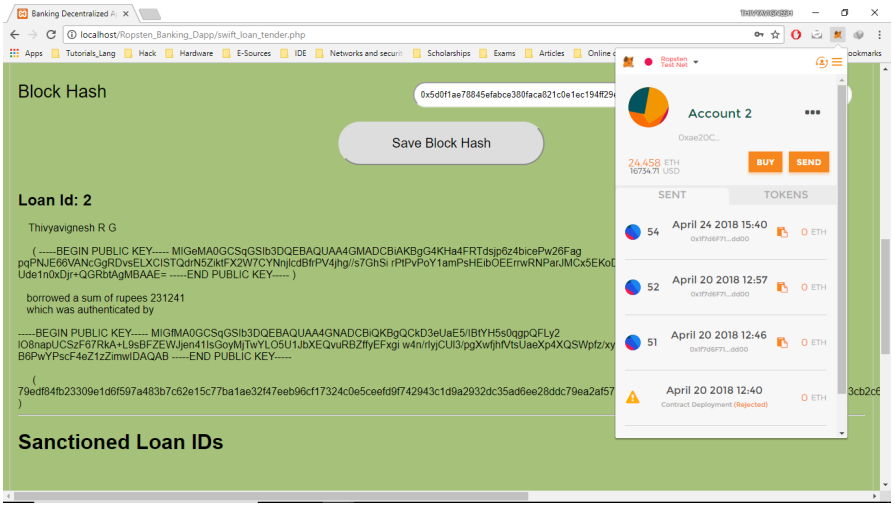

Figure 4.24: Loanee Details Added Successfully

Once the transaction is a success, the transaction information and the information of the block holding this transaction details can be viewed by clicking on the transaction in the metamask chrome extension. The abovementioned details for the setLoanee transaction are shown in figures 4.25 and 4.26.

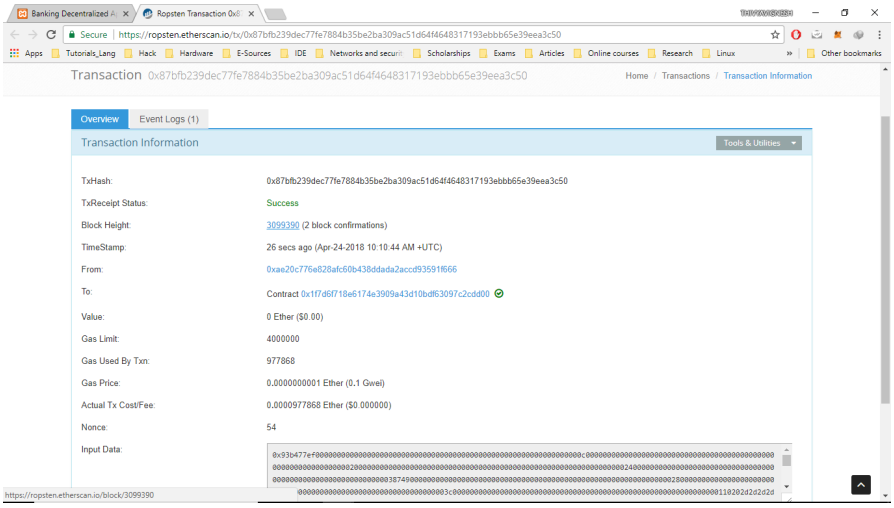

Figure 4.25: Transaction Information

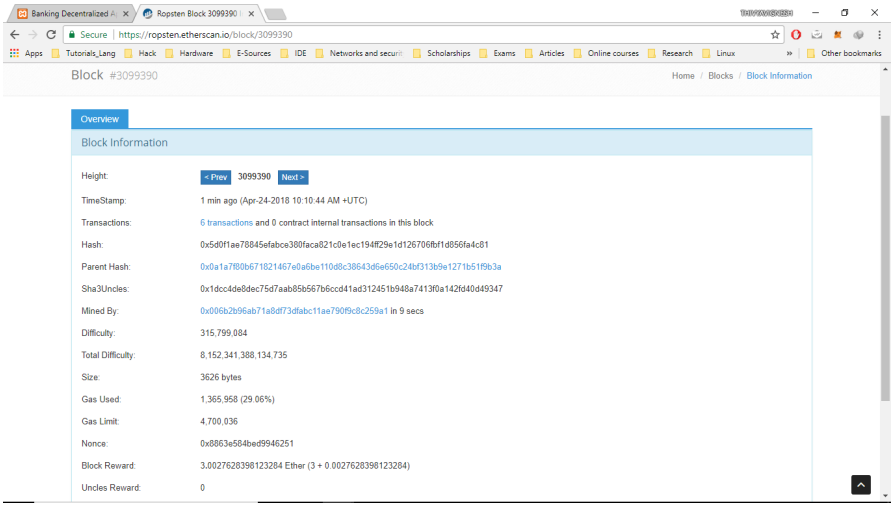

Figure 4.26: Block Information

Chapter 5

Conclusion

5.1 Technical Limitations of Block Chain

Though, Blockchain is one of the powerful trending technologies which has potential to change the framework of web entirely, it still has some technical limitations to consider as well. Fervent research work is being carried out to address many of these limitations. Some of the limitations of Blockchain are as follows.

Scalability problem in blockchain arise from the fact that blockchain is becoming longer and bulkier as days pass. As per the blockchain technology, there is a fixed block time interval which limits the rate of block entry in the blockchain. This, being the case delays the real time processing of millions of node. To address this issue, research is carried out in redesigning the blockchain as well as optimizing the storage features.

Mining Monopoly is possible when more miner nodes jointly operate to generate high revenue. The miners selfishy create a longer chain which can replace some extent of the original valid blockchain exploiting the longest chain acceptance feature of blockchain.

Privacy in blockchain cannot guarantee the transactional privacy since the values of all transactions and balances for each public key are publicly visible. Methods proposed to improve anonymity of blockchain has been categorized into Mixing and Anonymous.

5.2 Conclusion

Ethereum protocol, being the upgraded cryptocurrency not only focuses on transactions and mining, it also implements a nearly Turing-complete language on its blockchain, a prominent smart contract framework. This feature of Ethereum demonstrated the start of evolution from Web 2.0 to Web 3.0, the third generation of Internet-based services that collectively comprise what might be called the intelligent web, by incorporating the notion of decentralisation into web applications.

The appropriate approach is not to transform the entire banking system because blockchain is not a cure-all for all issues facing the banking system today. However, that being said, blockchain is an ideal technology to ensure the proof of integrity to data and reduce incidents of fraud. In a private permissioned blockchain, regulators, payment processors and auditors have real-time access to transactions, making it much easier to identify any attempted fraud or hack. A blockchain based platform mitigates fraud by establishing a network and securely sharing details about transactions between institutions in real time. The system allows institutions to maintain the privacy of valuable customer data while automatically detecting any elements of a fraud, small or large.

Moreover, the customer experience of banks in India is currently hindered by banks' ability to be technologically adapt to the numerous new digital initiatives being introduced. By integrating a blockchain technology solution with existing legacy systems, it can be easier, faster, and cheaper for banks to implement new digital solutions.

Bibliography

[1] I. Bashir. *Mastering Blockchain*. Packt Publishing, 2017.

[2] Blockchain. https://github.com/decrypto-org/blockchain-papers
https://github.com/Xel/Blockchain-stuff

[3] Ethereum project. https://www.ethereum.org/.

[4] Ropsten testnet. https://ropsten.etherscan.io/.

[5] Chris Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, Berkely, CA, USA, 1st edition, 2017.

[6] Web3. https://web3js.readthedocs.io/en/1.0/getting-started

[7] Ganache cli. https://www.npmjs.com/package/ganache-cli.

[8] Metamask. https://metamask.io/.

[9] Solidity. http://solidity.readthedocs.io/en/latest/index.html.

Appendix A

Solidity Codes

A.1 Back End : Solidity Code

The Back end code for the Banking Smart Contract is written in the language Solidity. The Solidity code written using the latest Solidity version 0.4.21 is explained here.

```
contract Owned {
    address owner;
    function Owned() public {
        owner = msg.sender;
    }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}
```

A base contract Owned is created with the features of identifying the owner of the contract. This ensures that the transaction will be processed only when the owner sends the transaction request message through Web3 API. This base contract is inherited by the contract Loan which comprises of the structures Loanee and Tend described below.

```
struct Loanee {
        string pkey;
        string fName;
        string lName;
        uint amt;
        string authkey;
        string authsign;
        uint id;
    }
mapping (uint => Loanee) Borrowers; string[] public LoaneeAccts;
event LoaneeInfo(
        string pkey,
        string fName,
        string lName,
        string authkey,
        string authsign,
```

```
        uint amt
    );
```

The struct Loanee defines the Loanee with the aforementioned state variables which holds the values of Public key of Loanee, loan details, bank authority public key and the authority signature verifying the details. The event LoaneeInfo is fired on the occasion of adding new loanees. The mapping Borrowers maps the loan id to the Loanee while the LoaneeAccts stores the public keys of loanees.

```
uint id; uint nextId = 0;
function setLoanee(string _address, string _fName, string _lName,
    uint _amt, string _authkey, string _authsign) public {
        var inst = Borrowers[nextId]; inst.pkey = _address;
        inst.fName = _fName; inst.lName = _lName;
        inst.amt = _amt; inst.authkey = _authkey; inst.authsign =
    _authsign;
        LoaneeAccts.push(_address) - 1; LoaneeInfo(_address, _fName,
    _lName, _authkey, _authsign, _amt);
        id = nextId; inst.id = id; nextId++;
    }
function getLoanee(uint id) view public returns (string, string,
    string, uint, string, string) {
        return (Borrowers[id].pkey, Borrowers[id].fName, Borrowers[id
    ].lName, Borrowers[id].amt, Borrowers[id].authkey, Borrowers[id].
    authsign);
    }
function countLoanee() view public returns (uint) { return
    LoaneeAccts.length; }
```

There are two functions (getLoanee, countLoanee) declared which read the state variables of Loanee. The function setLoanee gets the details from the web3 object as input parameters and sets the state variables.

```
struct Tend {
        string tendpkey;
        string org;
        uint tender_amt;
        uint tender_dur;
        uint tender_val;
        string tender_authkey;
        string tender_authsign;
    }
mapping (uint => Tend) Organisation; string[] public TendAccts;
event TendInfo(
        string tendpkey,
        string org,
        uint tender_dur,
        uint tender_amt,
        string tender_authkey,
        string tender_authsign
    );
```

The struct Tend defines the Tender with the aforementioned state variables which holds the values of Public key of the person who quotes the tender, tender details, bank authority public key and the authority signature verifying the details. The event TendInfo is fired on the occasion of adding new quotations. The mapping Organisation maps the tender id to the unique tender while the TendAccts stores the public keys of all the tenders.

```
uint nextTend = 0; uint tendid;
function setTend(string _address, string _org, uint _amt, uint _dur,
    string _authkey, string _authsign) public {
        var inst = Organisation[nextTend]; inst.tendpkey = _address;
        inst.org = _org; inst.tender_dur = _dur;
        inst.tender_amt = _amt; inst.tender_val = _dur * _amt;
        inst.tender_authkey = _authkey; inst.tender_authsign =
    _authsign;
        TendAccts.push(_address) - 1; TendInfo(_address, _org, _dur,
    _amt, _authkey, _authsign);
        tendid = nextTend; nextTend++;
    }
function getTend(uint id) view public returns (string, string, uint,
    uint, string, string) {
        return (Organisation[id].tendpkey, Organisation[id].org,
    Organisation[id].tender_dur, Organisation[id].tender_amt,
    Organisation[id].tender_authkey, Organisation[id].tender_authsign)
    ;
    }
```

Here, the function getTend declared reads the state variables of Tend, while the function setTend gets the details from the web3 object as input parameters and sets the state variables.

```
function minTender() view public returns (string, string, uint, uint,
    string, string) {
        if (TendAccts.length == max_Q) {
            uint j = 0;uint minT = Organisation[0].tender_val;
            for (uint i = 1; i < max_Q; i++) {
                if (Organisation[i].tender_val < minT) {
                    minT = Organisation[i].tender_val;j = i;
                }
            }
            return (Organisation[j].tendpkey, Organisation[j].org,
    Organisation[j].tender_amt, Organisation[j].tender_dur,
    Organisation[j].tender_authkey, Organisation[j].tender_authsign);
        }
        else { return ("NULL","NULL",0,0,"NULL","NULL"); }
    }
```

The function minTender returns the tender details of the tender with the minimum quotations. The duration of the tender and the cost quoted are considered in determining the minimum quotation.

Appendix B

Web3 and Javascript Codes

B.1 Setting Interface : Web3 Codes

The Web3 codes are the interface commands connecting the front end to the back end Solidity.

```
if (typeof web3 !== 'undefined') {
            web3 = new Web3(web3.currentProvider);
        } else {
            web3 = new Web3(new Web3.providers.HttpProvider("http://
    localhost:8545"));
        }
```

The above lines of the code check whether the Web3 provider is defined. The localhost is the Web3 provider used by us to connect to the Metamask Chrome Extension.The default account is the first blockchain node address which created the contract.

$web3.eth.defaultAccount = web3.eth.accounts[0];$

```
var PNBContract = web3.eth.contract(
        [{"anonymous": false,"inputs": [{"indexed": false,"name": "
    tendpkey","type": "string"},
        {"indexed": false,"name": "org","type": "string"},{"indexed":
     false,"name": "tender_dur","type":
        "uint256"},{"indexed": false,"name": "tender_amt","type": "
    uint256"},{"indexed": false,"name":
        "tender_authkey","type": "string"},{"indexed": false,"name":
    "tender_authsign","type":
        "string"}],"name": "TendInfo","type": "event"},{"constant":
    false,"inputs": [{"name":
        "_address","type": "string"},{"name": "_fName","type": "
    string"},{"name": "_lName","type":
        "string"},{"name": "_amt","type": "uint256"},{"name": "
    _authkey","type": "string"},{"name":
        "_authsign","type": "string"}],"name": "setLoanee","outputs":
     [],"payable":
        false,"stateMutability": "nonpayable","type": "function"},{"
    anonymous": false,"inputs":
        [{"indexed": false,"name": "pkey","type": "string"},{"indexed
    ": false,"name": "fName","type":
```

```
    "string"},{"indexed": false,"name": "lName","type": "string"
},{"indexed": false,"name":
    "authkey","type": "string"},{"indexed": false,"name": "
authsign","type": "string"},{"indexed":
    false,"name": "amt","type": "uint256"}],"name": "LoaneeInfo",
"type": "event"},{"constant":
    false,"inputs": [{"name": "_address","type": "string"},{"name
": "_org","type": "string"},
    {"name": "_amt","type": "uint256"},{"name": "_dur","type": "
uint256"},{"name":
    "_authkey","type": "string"},{"name": "_authsign","type": "
string"}],"name":
    "setTend","outputs": [],"payable": false,"stateMutability": "
nonpayable","type": "function"},
    {"constant": true,"inputs": [],"name": "countLoanee","outputs
": [{"name": "","type":
    "uint256"}],"payable": false,"stateMutability": "view","type"
: "function"},{"constant":
    true,"inputs": [],"name": "getendid","outputs": [{"name": "",
"type": "uint256"}],"payable":
    false,"stateMutability": "view","type": "function"},{"
constant": true,"inputs": [],"name":
    "getid","outputs": [{"name": "","type": "uint256"}],"payable"
: false,"stateMutability":
    "view","type": "function"},{"constant": true,"inputs": [{"
name": "id","type":
    "uint256"}],"name": "getLoanee","outputs": [{"name": "","type
": "string"},{"name": "","type":
    "string"},{"name": "","type": "string"},{"name": "","type": "
uint256"},{"name": "","type":
    "string"},{"name": "","type": "string"}],"payable": false,"
stateMutability": "view","type":
    "function"},{"constant": true,"inputs": [{"name": "id","type"
: "uint256"}],"name":
    "getTend","outputs": [{"name": "","type": "string"},{"name":
"","type": "string"},{"name":
    "","type": "uint256"},{"name": "","type": "uint256"},{"name":
 "","type": "string"},{"name":
    "","type": "string"}],"payable": false,"stateMutability": "
view","type": "function"},
    {"constant": true,"inputs": [{"name": "","type": "uint256"}],
"name": "LoaneeAccts","outputs":
    [{"name": "","type": "string"}],"payable": false,"
stateMutability": "view","type": "function"},
    {"constant": true,"inputs": [],"name": "minTender","outputs":
 [{"name": "","type": "string"},
    {"name": "","type": "string"},{"name": "","type": "uint256"
},{"name": "","type": "uint256"},
    {"name": "","type": "string"},{"name": "","type": "string"}],
"payable": false,"stateMutability":
    "view","type": "function"},{"constant": true,"inputs": [{"
name": "","type": "uint256"}],"name":
    "TendAccts","outputs": [{"name": "","type": "string"}],"
payable": false,"stateMutability":
    "view","type": "function"}]);
```

The Application Binary Interface (ABI) is used to encode the contracts to the Web3

object.

```
var PNB=PNBContract.at('0x1f7d6F718E6174e3909A43D10Bdf63097c2cdd00');
```

The address of the contract in the blockchain is passed as parameter to connect the javascript function calls to the contract.

B.2 Front End : Javascript Codes

The Javascript is the scripting language used to call the contract functions from the front end.

```
<script>
        var instEvent = PNB.LoaneeInfo({}, 'latest');

        instEvent.watch(function(error, result) {
            if (result) {
                if (result.blockHash != $("#insTrans").html())
                    $("#loader").hide();
                 document.getElementById("insTrans").value = result.
    blockHash;
                $("#instructor").html(result.args.fName + ' ' + result
    .args.lName
                +'<br><br>     ( '+result.
    args.pkey+'
                )<br><br>'+'   borrowed a sum of rupees
       ' + result.args.amt
                +'<br>   which was authenticated by<br>
    <br>'+ result.args.authkey
                +'<br><br>     ( '+result.
    args.authsign+' )');

        PNB.getid((err, res) => {
                  if (res)
                    $("#idloan").html('Loan Id: '+ res);
              });
            } else {
                 $("#loader").hide();
            }
        });

        $("#button").click(function() {
             $("#loader").show();
            PNB.setLoanee.sendTransaction(document.getElementById("
    address").getAttribute("placeholder"),
            $("#fName").val(), $("#lName").val(), $("#amt").val(),
            document.getElementById("authkey").getAttribute("
    placeholder"),document.getElementById("authsign").value,
            {from:web3.eth.accounts[0],
             gas:4000000}, (err, res) => {
                if (err) {
                    console.log("error in set!")
                    $("#loader").hide();
                }
```

```
            });
            ClearFields();
        });
</script>
```

The above script triggers the event LoaneeInfo whenever the setLoanee function is called. By doing so, we get a real time interface displaying the details entered by the user which is stored in the blockchain. The blockhash displayed is stored for future reference.

```
<script>
      $("#but").click(function() {
              $("#loader").show();
       PNB.getLoanee($("#add").val(),(err, res) => {
                  if (err) {
                      console.log("error in getLoanee!")
                      $("#loader").hide();
                  }
                  if (res) {
                $("#loanees").html('<b>Public Key:</b> '+res[0]+'<br>&
    nbsp;   <b>Name:</b>
                '+ res[1] +' '+ res[2] + '<br>    <b
    >Amt:</b> ' + res[3]
                +'<br>    <b>Authenticator:</b> '+
    res[4]
                +'<br>    <b>Authenticator Sign:</b>
     '+res[5]);
                  }
              });
          $("#loader").hide();
        });
</script>
```

Here the function getLoanee is called to display the details of the loan id entered by the admin.

```
<script>
        $("#tender_button").click(function() {
            PNB.setTend.sendTransaction(document.getElementById("
    tender_address").getAttribute("placeholder"),
            $("#orgname").val(), $("#tender_duration").val(), $("#
    tender_amt").val(),
            document.getElementById("tender_authkey").getAttribute("
    placeholder"),
            document.getElementById("tender_authsign").value, {
             from:web3.eth.accounts[0],
             gas:4000000}, (err, res) => {
                if (err) {
                    console.log("error in set!")

                }
            });
            ClearFields();
        });
```

```
</script>
```

Similar to setLoanee, the setTend function is called using function sendTransaction().

```
<script>
     $("#tender_but").click(function() {
            $("#loader").show();
      PNB.minTender((err, res) => {
                if (err) {
                    console.log("error in minTender!");
                    $("#loader").hide();
                }
                if (res) {
                 $("#tenderwinner").html('<b>Public Key:</b> '+res
    [0]+'<br><b>Organisation:</b> '+ res[1]
                 + '<br><b>Amt(Rs):</b> ' + res[2]+'<br><b>Duration(
    Months):</b> ' + res[3]
                 +'<br><b>Authenticator:</b> '+res[4]+'<br><b>
    Authenticator Sign:</b> '+res[5]);
                }
            });
       $("#loader").hide();
        });
</script>
```

The results of the minTender function is displayed when asked by the user.

```
<script type="text/javascript">
  function generatesignature()
  {
    var msg=document.getElementById("address").value + document.
    getElementById("fName").value
    +document.getElementById("lName").value + document.getElementById
    ("amt").value +
    document.getElementById("authkey").value  ;
    var privatek=document.getElementById("privatekey").value;
    var rsa = new RSAKey();
      rsa.readPrivateKeyFromPEMString(privatek);
      var hashAlg = 'sha1';
      var hSig = rsa.sign(msg, hashAlg);
      signature=linebrk(hSig, 64);
      document.getElementById("authsign").value = linebrk(hSig, 64);
}
</script>
```

The RSA signature scheme used is called whenever the admin decides to sign the loanee or the tender details for authentication.