WEB-ENABLED SECURED LOAN SANCTION APPLICATION USING SMART CONTRACT WITH BLOCKCHAIN TECHNOLOGY

KAVYA S
(20PCA007)

Project Submitted

In partial fulfillment of the requirements for the Award of

Master’s Degree in Computer Applications

DEPARTMENT OF COMPUTER SCIENCE

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND

HIGHER EDUCATION FOR WOMEN

COIMBATORE – 641043

MAY – 2022

WEB-ENABLED SECURED LOAN SANCTION APPLICATION USING SMART CONTRACT WITH BLOCKCHAIN TECHNOLOGY

KAVYA S

(20PCA007)

Project Submitted

In partial fulfillment of the requirements for the Award of

Master's Degree in Computer Applications

DEPARTMENT OF COMPUTER SCIENCE

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND HIGHER EDUCATION FOR WOMEN

COIMBATORE – 641043

MAY – 2022

Signature of the Head of the Department **Signature of the Supervisor**

Viva-voce Examination held on _______________________

Signature of the Examiner

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I would like to express my sincere thanks to God Almighty, for his constant love and grace that he showered upon me, which kept me in good health, and sound mind without which myproject would not have reached a successful end.

I would like to express my deep sense of reverential gratitude to the former Chancellor late **Dr.P.R.Krishnakumar Ji**, for providing all the facilities during my study.

I would like to express my deep sense of reverential gratitude and sincere thanks to **Dr.S.P.Thyagarajan,** Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing all the facilities during my study.

I owe my great deal of gratitude to **Dr. V Bharathi Harishankar M.Sc., M.Ed., Dip. Spl. Edn., M.Phil., Ph.D.,** Vice-Chancellor, for extending all resources that facilitated the conduct of the project study.

I express my gratitude to **Dr. S. Kowsalya, Registrar, M.Sc., M.Phil., Ph.D.** for providing all the necessary facilities for the project.

I am also thankful to former Dean, **Dr. (Mrs.) Udaya Chandrika**, **M.Sc., M.Phil., Ph.D.,** School of Physical Sciences and Computational Sciences, for granting the facility required.

I am also thankful to **Dr. G. Padmavathi, M.Sc., M.Phil., Ph.D.,** Dean, School of PhysicalSciences and Computational Sciences, for granting the facility required.

I wish to place my deep sense of gratitude to **Dr. (Mrs.) Vasantha Kalyani DavidM.Sc. M.Phil. (Maths), M.Phil. (Computer Science), Ph.D.** Head, Department of Computer Science, for the support and encouragement to complete the project.

I heartily thank my project guide **Dr.(Mrs.) G.GEETHA B.E., MTech M.Phil., Ph.D.**, Senior Technical Assistant of Computer Science Department , for imparting the tremendous knowledge and well-timed support for the successful completion of my project. Her guidance and constant supervision helped me in completing the project in time.

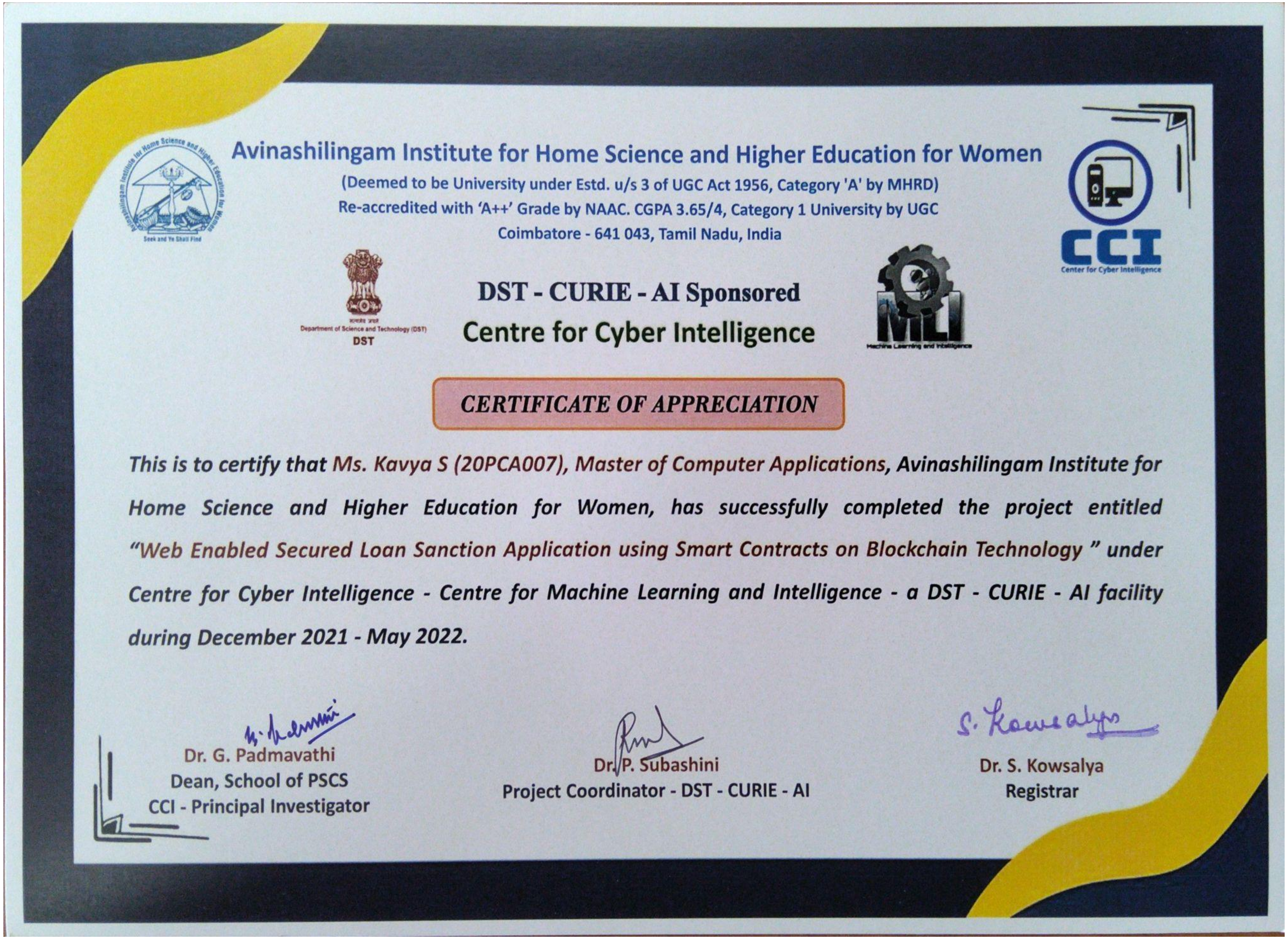
I would like to acknowledge the help rendered by Center for Cyber Intelligence, DST – CURIE – AI Sponsored Phase II for providing the laboratory facilities to execute my project.

.

I have great pleasure in expressing my deep sense of gratitude to all the teaching and non-teaching staff members who stood behind the screen for the completion of the project.

Finally, I would like to thank my parents, family members, friends, and all well-wishers for their kind inspiration, blessings, and encouragement during the project time.

CERTIFICATE



Avinashilingam Institute for Home Science and Higher Education for Women

(Deemed to be University under Estd. u/s 3 of UGC Act 1956, Category 'A' by MHRD)
Re-accredited with 'A++' Grade by NAAC. CGPA 3.65/4, Category 1 University by UGC
Coimbatore - 641 043, Tamil Nadu, India

Department of Science and Technology (DST)
DST

CCI
Center for Cyber Intelligence

Machine Learning and Intelligence

DST - CURIE - AI Sponsored
Centre for Cyber Intelligence

CERTIFICATE OF APPRECIATION

This is to certify that Ms. Kavya S (20PCA007), Master of Computer Applications, Avinashilingam Institute for Home Science and Higher Education for Women, has successfully completed the project entitled "Web Enabled Secured Loan Sanction Application using Smart Contracts on Blockchain Technology " under Centre for Cyber Intelligence - Centre for Machine Learning and Intelligence - a DST - CURIE - AI facility during December 2021 - May 2022.

Dr. G. Padmavathi
Dean, School of PSCS
CCI - Principal Investigator

Dr. P. Subashini
Project Coordinator - DST - CURIE - AI

S. Kowsalya
Dr. S. Kowsalya
Registrar

ABSTRACT

ABSTRACT

Blockchain is a decentralized ledger that is used to securely trade digital currency, as well as to conduct and handle transactions. Banking systems can migrate from their current methods to a digital, immutable, and distributed ledger that Blockchain can provide. The necessity for collateral, the time necessary for settlements, currency denomination variations, third-party mediation, and other issues all complicate Loan transactions. Fraudsters are particularly interested in multi-step procedures that need human participation. Information may be transmitted in real-time with blockchain, and the ledger can only be altered with the agreement of all parties. This can help save time, money, and the potential for fraud. It's also less probable that a party won't be paid if the completion time is shortened.

This project deals with the development of an online transaction application usingGanache and Metamask Wallet by the means of Ethereum as a digital currency. The Process of Web-Enabled Loan Sanction Application using Smart Contract with Blockchain Technology comprises eight phases. In phase 1, a metamask wallet extension is created to add Ethereum. Phase 2 deals with collecting Ethereum (digital currency) in the form of tokens to activate metamask Wallet. Phase 3 involves the initialization of Ganache Software to show the transact from one account to another account using metamask wallet. In Phase 4, a linkage between Ganache software with metamask wallet is created. In Phase 5, a web application is designed using HTML, CSS, JAVASCRIPT, PHP, and MySQL to prevent fraudulent attacks on sanctioning of loans through the decentralized process. In phase 6, smart contracts are created using Solidity programming environment and are deployed in the backend, which runs on Ethereum simulator Ganache and Ropsten testnet. In Phase 7, in order to test the smart contract Remix IDE desktop application is used to validate the smart contract to ensure authentication security. In Phase 8, finally, a decentralized banking application is developed using Ethereum with the support of metamask chrome extension have been done efficiently using Blockchain Technology.

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

CONTENT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

BLOCKCHAIN

Chapter 1

Blockchain

1.1 Introduction

Blockchain technology has built the backbone of a new sort of internet by allowing digital information to be distributed but not replicated. Consider a spreadsheet that has been copied thousands of times over a network of computers.

This is due to frequent systemic Failures that detect human errors. Blockchain technology is the greatest solution for this issue. It is surprisingly common for an informal settlement mechanism like Swift to be on an isolated ledger from the payment settlement mechanism. If the banks use a ledger that stores information settlement distributed across all the participants, then the fraudulent user may reflect on all the available participants in the transactions, auditors, and regulators.

Figure 1.1: Blockchain

A blockchain is typically managed as a distributed ledger, as shown in 1.1, 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 be changed retroactively without affecting all subsequent blocks, which requires network majority agreement. When a group of people share something, the distributed part comes into play. Consider how many legal documents should be used in this manner. Why can't all business documents be shared instead of transferred back and forth, instead of being passed to each other, losing track of versions, and being out of sync with the other version? So many different types of legal contracts would be ideal.

1.2 Structure

Blockchains have the potential to increase trust in digital data. It is nearly impossible to removeor change information once it has been written into a blockchain database.

Block: It is a collection of transactions recorded in a ledger over a specific time period. Every blockchain has a different block size, period, and triggering event. Not all blockchains' primary goal is to record and secure a record of their cryptocurrency's movement.

Hash: A hash that connects one block to another by mathematically "chaining" them. This is one of the most difficult blockchain concepts to grasp. It's also the glue that holds blockchains together and enables them to create mathematical trust.

Network: The network is made up of "complete nodes." Consider them to be the computer that is running an algorithm that is securing the network. Each node keeps a complete record of all transactions ever recorded in that blockchain.

1.3 Main Characteristics

Distributed: The term "distributed" refers to the fact that Blockchain may run on anycomputer provided by volunteers all over the world. Because there is no central database and no authority to track it, it cannot be shut down or hacked. Any digital transaction may take place directly between two parties, with no need for an intermediary**.**

Encrypted: The days of firewalls are over; each block of the blockchain is extensively encrypted, using both public and private keys. Encryption aids in the security and preservation of each block's individuality.

Inclusive: An international transaction taking place without the use of paperwork or legal channels. Satoshi, envisioned a streamlined payment verification mode that could be used on a mobile device without the requirement for legal documents.

Immutable: Once a transaction has been initiated, verified, and validated by others,it is added to the blockchain as a new block with a timestamp and linked to the previous block with a unique signature. It cannot be updated or altered and hence remains in the ledger indefinitely.

Public: In some sense, blockchain is public in nature, as it can be viewed by anybodyon the network. This makes transactions clear and open; no transactions are hidden.

Historical: In some sense, blockchain is historical Someone can't easily steal a blockor bitcoin. this makes the transactions so clear and has been recorded historically.

1.4 Consensus Model

Blockchains are distributed systems that share a common state; the network must agree onthe distributed ledger's content. As a result, each blockchain requires a consensus model. Itguarantees that the following block in the chain has the sole version of the truth. The use of a consensus method allows the blockchain to circumvent the need for a central authorityto keep track of all accounts. Consensus models must have some fundamental qualities to order to be useful in blockchain implementation.

Aliveness: A consensus process ensures that all nodes involved finally produce a result.

Consistency: If all nodes deliver the same valid output, a consensus process is considered safe.

Fault Tolerance: If a consensus system can recover from a node failure, it is said to be fault-tolerant.

Figure 1.2 depicts the level of security provided by bitcoin's data storage. The modelbecomes even more secure as a result of the consensus models.

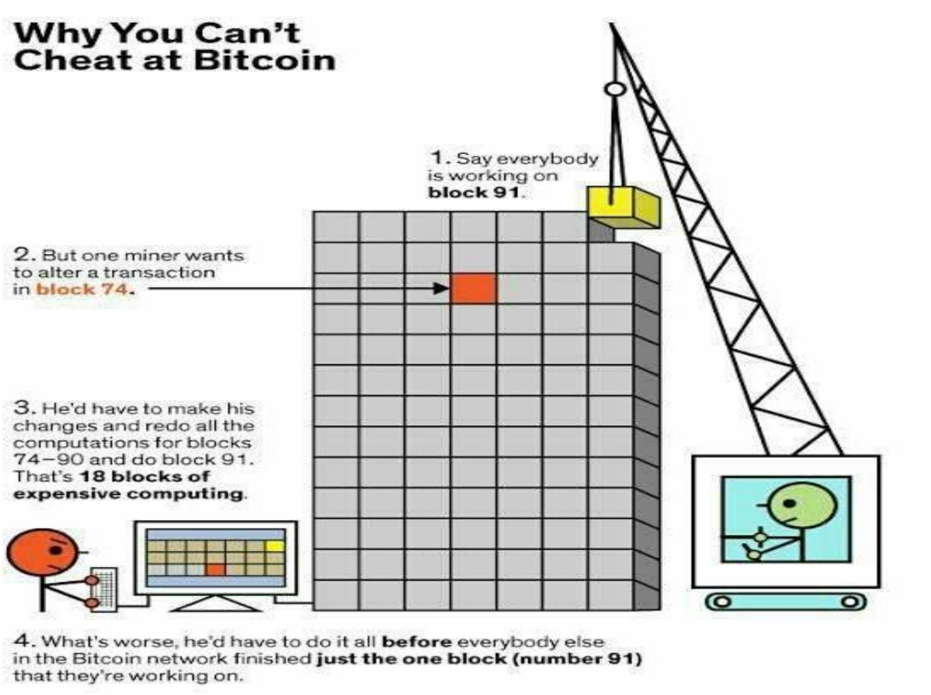

Why You Can't Cheat at Bitcoin
1. Say everybody is working on block 91.
2. But one miner wants to alter a transaction in block 74.
3. He'd have to make his changes and redo all the computations for blocks 74–90 and do block 91. That's 18 blocks of expensive computing.
4. What's worse, he'd have to do it all before everybody else in the Bitcoin network finished just the one block (number 91) that they're working on.

Figure 1.2: Consensus

1.4.1 Proof of Work

This is the most widely used algorithm, which is employed by currencies like Bitcoin and Ethereum, each with its unique set of features. The performer with the most processing power is the one who will solve the aforementioned problem first the majority of the time. These characters are also known as miners. Its widespread success can be attributed to the following characteristics:

- It is difficult to find a solution to the provided challenge.

- When provided a solution to a problem, it is simple to confirm that it is right.

- When a new block is mined, the miner is compensated with some cash (block reward, transaction fees), incentivizing him or her to continue mining. Other nodes verify the authenticity of the block in Proof of Work by ensuring that the hash of the block's data is less than a predefined value.

1.4.2 Proof of Authority

Proof of Authority (PoA) is an alternative consensus method that does not rely on nodes solving arbitrarily tough mathematical problems, but rather on a set of "authorities" . Nodes that are explicitly permitted to produce new blocks and protect the blockchain. The three basic factors that must be met in order for a validator to be formed are as follows:

• Identity must be formally validated on-chain, with the option of cross-checking the data in a publicly accessible domain.

• Eligibility must be difficult to attain in order for the blocks obtained and valued to be validated. (For instance, potential validators must have a public notary license.)

• The checks and methods for establishing authority must be completely uniform.

1.4.3 Proof of Stake

Proof of Stake eliminates the energy and processing power requirements of PoW and substitutes them with stake. Stake refers to the amount of cash that an performer is prepared to lock up for a certain period of time. In exchange, individuals receive a chance proportional to their share to be the next leader and pick the next block. Next and Blackcoin are two existent currencies that employ 100% PoS. The fundamental difficulty with PoS is the so-called "nothing-at-stake" dilemma. In the case of a fork, stakes are not disincentivized from staking in both chains, and the risk of double-spending increases.

CHAPTER 2
ETHEREUM

Chapter 2

Ethereum

2.1 Introduction

Ethereum is a blockchain-based distributed computing platform and operating system that is open source and public. It has smart contract (scripting) features. Through transaction-based state transitions, it supports a modified version of the Nakamoto consensus.

Instead of mining for bitcoin, miners on the Ethereum blockchain labor to earn Ether, a form of crypto asset that powers the network. Ether is used by application developers to pay for transaction fees and services on the Ethereum network, in addition to being a tradeable coin.

2.2 Terminologies

2.2.1 Address

The prefix "0x," a common identifier for hexadecimal, is concatenated with the rightmost 20 bytes of the Keccak-256 hash (big endian) of the ECDSA public key to form Ethereum addresses. A byte in hexadecimal is represented by two digits, therefore addresses have 40 hexadecimal digits. (For example, 0xb794F5eA0ba39494cE839613fffBA74279579268 from Poloniex.)

2.2.2 Ethereum VM

The Ethereum Virtual Machine focuses on providing security and allowing machines all aroundthe world to execute untrusted programs. To be more particular, this project aims to combat DDoS attacks, which have grown increasingly widespread in the cryptocurrency industry. Furthermore, the EVM assures that programs do not have access to each other's state, guaranteeing that communication may take place without interruption.

To put it another way, the Ethereum Virtual Machine is intended to serve as a runtime environment for Ethereum-based smart contracts. Smart contracts are highly popular these days, as most cryptocurrency aficionados are aware. On the Ethereum blockchain, this technology can be used to automatically complete transactions or perform specific operations. Smart contracts are expected to transform finance and other industries in the next years, according to many experts.

2.2.3 Gas

The term "gas" refers to a special Ethereum unit. It estimates how much "work" an action or group of actions takes: for example, calculating a Keccak256 cryptographic hash will cost 30 gas each time it is calculated, plus an additional 6 gas for every 256 bits of data hashed. Every transaction or contract on the Ethereum platform can conduct a fixed amount of operations, with operations requiring more computing resources costing more gas than operations requiring few computational resources.

Gas is vital because it ensures that the right fee is paid by transactions submitted to the network. By requiring a transaction to pay for each operation it performs (or causes a contract to perform), we ensure that the network does not become clogged with work that isn't useful to anyone. This is different from the Bitcoin transaction fee, which is dependent only on the transaction's size in kilobytes. Because Ethereum enables the execution of arbitrarily complicated computer code, a little piece of code can accomplish a lot of computational effort. As a result, instead of establishing a price based on the length of a transaction or contract, it's critical to quantify the labor done immediately.

2.2.4 Ether

Ether is a key coin for Ethereum's operation, which includes a public distributed ledger for transactions. It's a unit of computation used in transactions and other state transitions, and it's used to pay for gas.

2.3 Ethereum vs Bitcoin

Ether differs from Bitcoin in a number of ways:

- Its block time is 14 to 15 seconds, compared to bitcoin's 10 minutes

- Mining ether generates new coins at a generally stable rate, which may vary during hard forks, whereas mining bitcoin generates new coins at a rate that halves every four years.

- It employs the Ethash algorithm for proof-of-work, reducing the advantage of specialist ASICs in mining.

- Transaction fees vary according to computational difficulty, bandwidth usage, and storage requirements (in a mechanism called as gas), whereas bitcoin transactions compete based on transaction size (in bytes).

- The price of each Ethereum gas unit can be defined in a transaction. This is usually expressed in Gwei. Bitcoin transaction fees are often indicated in Satoshi's per byte.

- Transaction fees for ether are often much cheaper than those for Bitcoin. In December 2017, the median transaction fee for ether was $0.33, while the fee for bitcoin was $23.

- Unlike Bitcoin's UTXO system, which is more similar to spending currency and receiving change in return, Ethereum employs an accounting system in which Wei values are debited from one account and credited to another. In terms of storage space, complexity, and security/anonymity, both methods offer advantages and disadvantages.

2.4 Testnets

Testnets are Ethereum blockchain replicas that are nearly identical to the Mainnet except that their Ether is worthless. There are three different types of testnets, as explained below.

2.4.1 Public Test

Everyone can use public testnets because they are connected to the internet. Anyone, includingpopular wallet interfaces like My Ether Wallet and MetaMask, can connect to them at any moment.

Ropsten

Ropsten was released in November of 2016. Its Ether can be mined in the same way that it canon the Mainnet. It's supported by both Geth and Parity, two separate versions of the Ethereumnode software. Ropsten is mostly like the current Mainnet of the three testnets. Because its consensus process is PoW (i.e., it can be mined), its outcomes are similar to those of the Mainnet, making the simulation of transaction confirmations the most realistic.

The current blockchain file size for Ropsten is roughly 9 GB. Ether can be mined or demanded on the Ropsten network using the Ropsten Faucet, a website dedicated solely to giving away free test Ether. Because Ether may be mined on Ropsten, it is vulnerable to spam attacks, which overload the network with worthless transactions. It's simple to produce this deluge if Ether is free and easy to obtain. In February 2017, attackers mined massive quantities of Ether and continued to send overly large transactions through the network. Because Ethereum's block size limit is designed to be flexible and rise with demand, they were able to increase it to several billion units of gas from the previous 4 million.

It's worth noting that Ropsten only differs from the Mainnet (where we all keep our"real" Ether) by agreement. Ropsten's Ether was worthless, and it is now. Ropsten has its ownmining pools, software, and so forth, but we all agreed that its Ether has no value, thus it doesn't.Here, we see how the community's consensus defines the value of an asset – or, more precisely,the lack of value of this asset.

Rinkeby

Launched by the Ethereum team in April 2017, Rinkeby shares the benefits of Kovan with twosmall differences: it does not support Parity and only works with Geth, and it employs a slightly different PoA consensus process. Etherscan also supports Rinkeby: https://rinkeby.etherscan.io/ Rinkeby An authorised faucet can be used to request ether.

2.4.2 Private Test

A private test network is analogous of own personal blockchain, or own copy of Ethereum. When launching a private blockchain, a Genesis file must be created from which a tool like Geth may construct the new chain. This chain is then examined and interacted with using programmes such as Mist, MetaMask, MyEtherWallet, and others.

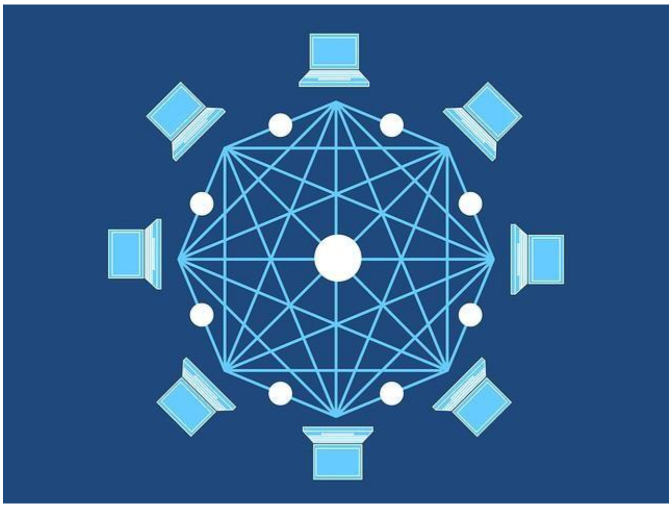


Figure 2.1 Private Testnet

Private testnets are ideal for cooperation and controlled situations where mining and transactionconfirmations must be simulated without exposing their network to the outside world and risking spam attacks. There is no cost to creating one, other than a small portion of the developer's computer's CPU and disc space being used while the testnet is active. When a private testnet has grown sufficiently, it can be made available to the public through the internet, where additional interested parties can join to it and enlarge it. This is ideal for experimentation, collaboration, cross-application interaction, and other uses. When running a private testnet, Ether may be mined on even the most basic computers, and some addresses can even have some Ether pre-mined for future usage during activation.

2.4.3 TestRPC

TestRpc is a NodeJS package that runs on a single machine and simulates the Ethereum network. It will generate many Ethereum addresses when it launches, each with some Ether already on it. Though a nice concept in theory, Testrpc frequently fails in practise due to faults, making for an especially stressful experience. In our experience, it's quicker and more dependable to simply establish a private testnet with all the bells and whistles of a true blockchain, as shown in figure 2.1, rather than bothering with Testrpc.

2.5 Smart Contracts

Smart contracts allows to exchange money, property, shares, or anything else of value in a transparent, conflict-free manner without the use of an intermediary or whatever is deposited into an account. More specifically, smart contracts (Figure 2.2) not only establish the rules and penalties around an agreement in the same way that traditional contracts do, but they also automatically enforce those responsibilities.

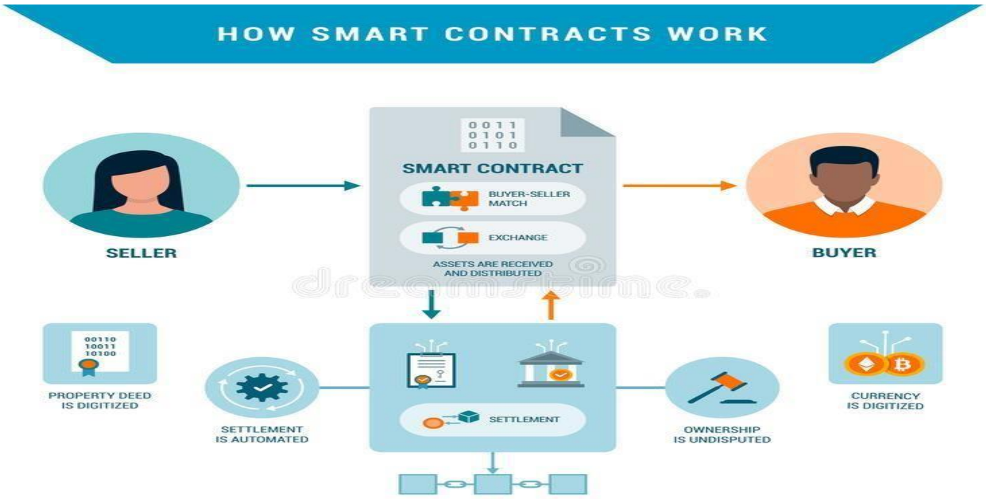


Figure 2.2: How Smart Contract Works

Smart contracts can be used in a variety of contexts, including financial derivatives, insurance premiums, breach contracts, property law, credit enforcement, financialservices, legal processes, and crowdfunding agreements.

2.6 Ethereum Wallet

A cryptocurrency wallet is a piece of software that maintains private and public keys and interacts with various blockchains to allow users to transfer and receive digital money while also monitoring their balance.

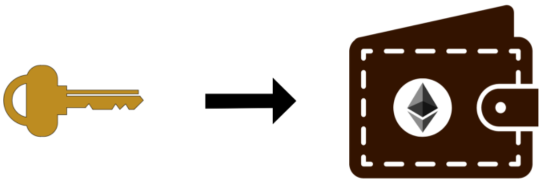


Figure 2.3: Wallet

The Ethereum Wallet (Figure 2.3) provides a portal to decentralised apps on the Ethereum network. It enables to store and safeguard ether and other Ethereum-based crypto-assets, as well as design, implement, and use smart contracts. Millions of people use bitcoin wallets, but there is a lot of misinformation about how they work. Digital wallets, unlike traditional "pocket" wallets, do not store currency. In truth, currencies are not held in a single location or exist in any physical form whatsoever. All that exists are records of transactions kept on the blockchain.

CHAPTER 3
SYSTEM CONFIGURATION

Chapter 3

System Configuration

3.1 HARDWARE REQUIREMENT

RAM	:	4GB and Higher
Processor	:	Intel i3
Disk	:	500GB

3.2 SOFTWARE REQUIREMENT

Operating System	:	Windows10
Front end	:	WEB3, HTML, CSS, JAVASCRIPT.
Back end	:	REMIX IDE(Solidity), PHP, NODE.JS, Metamask, Ganache-Cli,

3.3 ABOUT THE SOFTWARE

3.3.1 Web 2.0 vs Web 3.0

Table 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 lists the tools used in centralized networks (Web 2.0), and decentralized networks (Web 3.0), and their current transition status.

a) Architecture of Ethereum

The Architecture of Ethereum is shown in figure 3.1. The consensus layer isa protocol that describes the ledger format as well as a consensus function that can be used to determine which of the multiple candidate ledgers is theconsensus ledger. The economic layer contributes to the creation of tokens that incentivize nodes to conduct computation and other particular functions.Blockchain services include the programs or codes that carry out the process. The interoperability layer contains exchange protocols that are responsible for sharing information between network nodes and utilizing exchanged messages. Browsers are used to access decentralized applications.

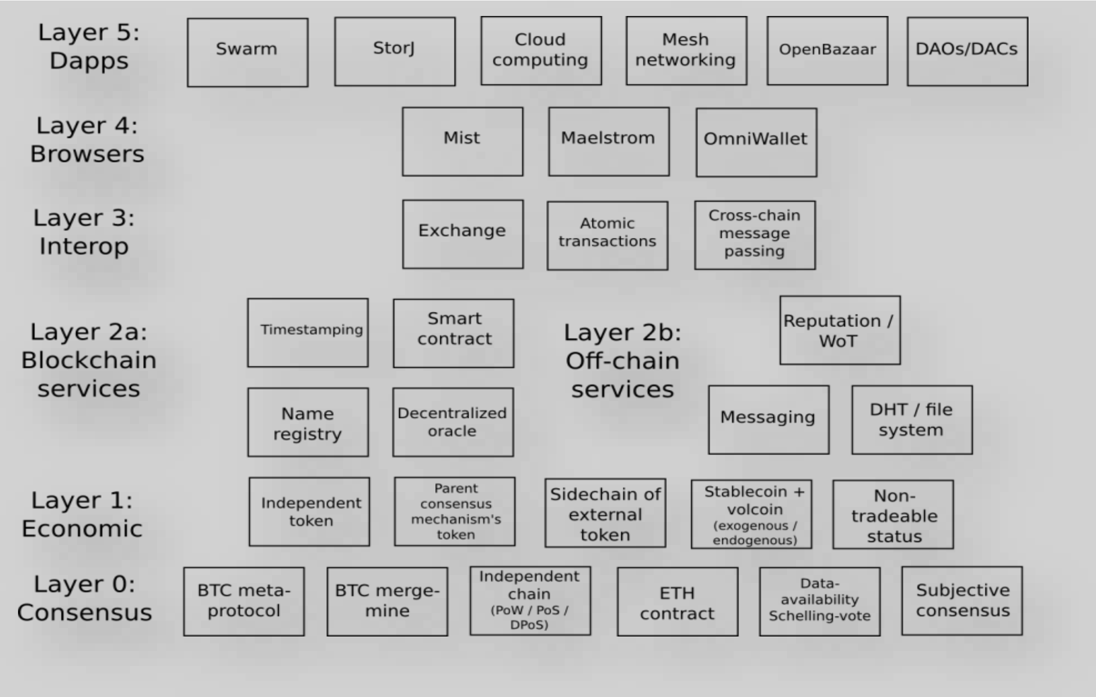


Figure 3.1: Architecture of Ethereum

3.3.2 Ethereum Client

The Ethereum network is made up of Ethereum nodes that are linked together. Ethereum nodes send and receive blocks of data from their peers, as well as perform validation and mining. Decentralized apps communicate with Ethereumnodes in order to transmit transactions. Transactions can be used to transfer ethe or to install or execute contracts. The Ethereum client reference implementation is available in a variety of languages, including C++ (eth), Python (pyethapp), and Golang (geth). There are also some third-party implementations available.

a) Geth

Geth is a Golang version of the Ethereum client (Fig. 3.2). It is a command-line utility. There are several commands in geth that are used to manage and run a full ethereum node. The generalsyntax of geth use is as follows: [options] command [command options] [args] Geth.

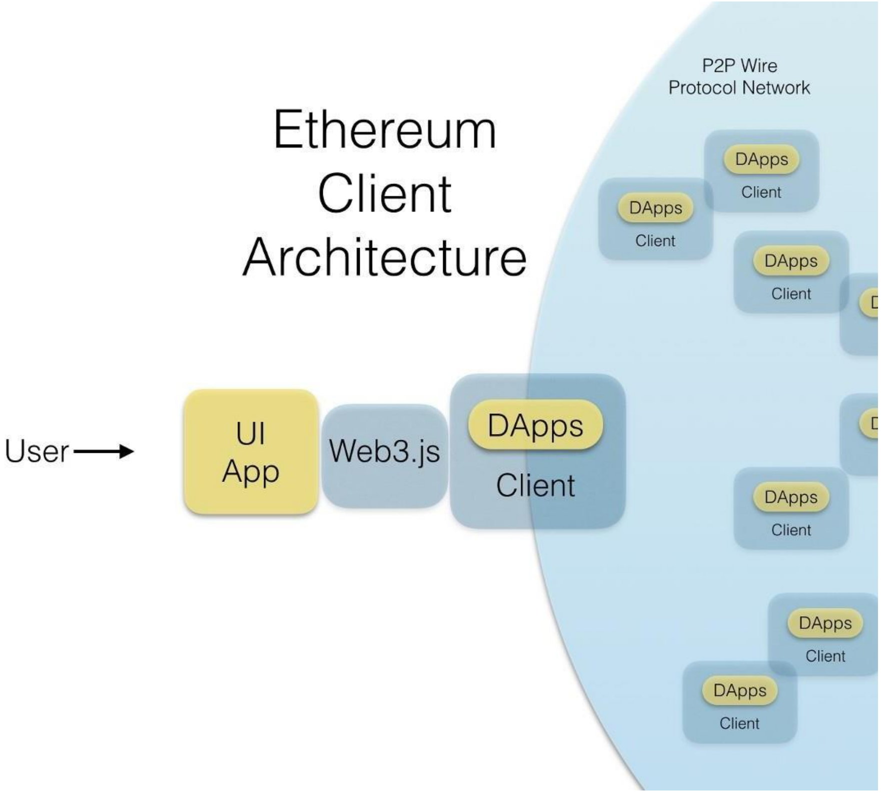


Figure 3.2: Ethereum Client Architecture

3.3.3 Web Javascript API

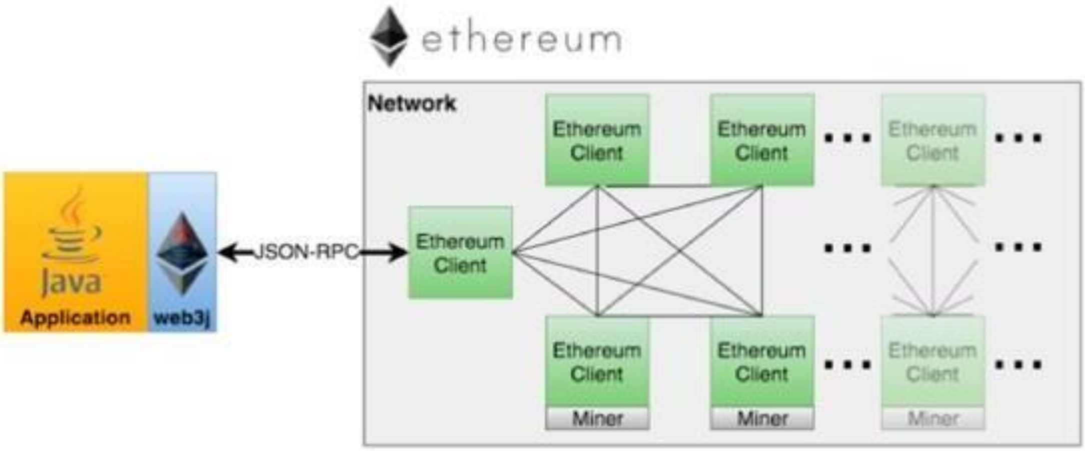


Figure 3.3: Web3J

Decentralized applications must communicate with Ethereum nodes. There are several libraries available to make connecting to Ethereum nodes easier. Web3 JSas illustrated in figure 3.3 is the javascript library. Ethereum clients support Javascript run time environment and web3 API. Geth, in addition to the aforementioned APIs, also includes management APIs, which are used for maintaining and controlling nodes. (Web3 applications) can be interactive as wellas non-interactive. Methods in web3 API and management API are classified as objects based on the functions they provide. Shorthand’s for APIs are also available.

Table 3.2: Web3

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

Table 3.3: Management

Admin	Node Management
Personal	Account management
Miner	Miner Management

As illustrated in figure 2, Web3 java script APIs serve as an interface between the front end of the decentralized application and its backend, which contains the smart contracts. Compilation with Web3 API only supported till Geth version 1.5.9. The function Web3.eth.compile.solidity(string,callbackfunc) is used to compile contracts.

3.3.4 Frameworks

a) Remix

The remix is the browser-based development environment for Ethereum contracts as shown in figure 3 (correct number).It includes an integrated compiler and Solidity runtime environment. It provides a basic user interface for drafting and maintaining contracts. The compilation and deployment processes are not automated. Three ways to inject a blockchain into a browser are using a Javascript Virtual machine (JVM), connecting to a web3 provider, and using an injected web3 instance (like metamask).

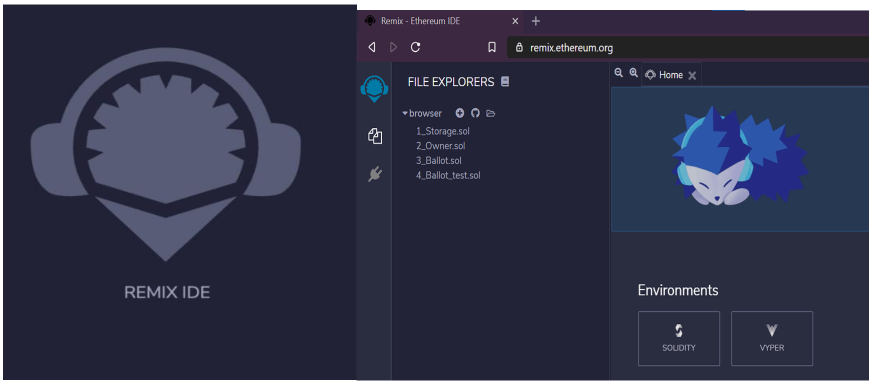


Figure 3.4 Remix IDE

b) Ethereum Simulators

Ethereum simulators are open source tools used to test smart contracts during the development process. They build networks that are identical to Ethereum networksbut are not linked to real-time Ethereum nodes. All executions take place in the simulator's sandbox. There is no mining involved here. As a result, it is a more efficient method of deploying and testing smart contracts throughout the development phase.

i) Ganache

Ganache is a personal blockchain engine and an ethereum emulator that is part of the truffle framework. Ganache is accessible in both command line and graphical user interface forms. Ganache CLI serves as an alternative for testRPC. Ganache CLI is used for tools and automation. Working, debugging, and configuringconfigurations are simplified in Ganache GUI. It includes a built-in block explorerfor seeing information about newly produced blocks and transactions.

Figure 3.5 GANACHE

c) Metamask

Metamask is a browser plugin that allows user to run Ethereum decentralized applications in browser without having to run a full Ethereum node. It injects web3API into the Javascript context of every webpage, allowing decentralized applications to be read from the blockchain. It also allows users to create and manage their own identities, so that when a Dapp wishes to make a transaction and write to blockchain, the user receives a secure interface to examine thetransaction before accepting or rejecting it.

Figure 3.6 METAMASK

3.3.5 Solidity

Smart contracts can be created in a variety of languages. It is possible to write it inserpentine, list, or solidity. The popular language for contract writing is Solidity.

a) Variables

Solidity is a static language, variable types are defined at build time. Some of the most common variable types include Booleans, integers, addresses, and fixed-size arrays. Combining many elementary types yields complex types.

```
Address    x = 0x123 ;

uint128    a =    1;
```

b) Arrays

Solidity can handle both storage and memory arrays. The size of a solidity array might be fixed or dynamic. Bytes and string data types are used in special arrays. For storage arrays, the element type is arbitrary; for memory arrays, it is not mapping.

```
Contract Simple {

unit [ ]  memory a = new        uint [ ] ( 7 ) ;
bytes memory b = newbytes ( len ) ;
}
```

c) Structs

In solidity, new types in the form of structs can be defined. Structs can contain all of thestandard data type variables, as well as arrays and maps, but not members of their own type.

```
contract SampleFunding{

struct Funder{
address addr ;

uintamount ;

}

}
```

d) Functions

There are two types of functions in solidity: internal and external. Internal functions are those that can only be called within the current contract. Internal functions are default functions.

```
Function (<parameter types >) { internal | external } [ pure | constant | view |

     payable ][ returns (<return types >)]
```

e) Mappings

Mappings are defined using the syntax mapping(keytype =>valuetype). Any supported data type, including mapping and transformations, can be used as a Valuetype. A contract, struct, enum, dynamically sized array, and mapping are the only types that cannot be used as keytypes. Mappings are similar to a hash table in that each key corresponds to a value.

```
contract MappingSample{

      mapping ( address =>uint ) public      balances ;

     function update ( uintnewBalance ) public {

     balances [msg .sender ] = newBalance ;

     }

}
```

f) Contract Deployment

When it comes to implementing smart contracts on the Ethereum network, there are twocomponents:

- Bytecode
- Abi Definition

Bytecode is the binary that is deployed in the network. The application binary interface is a json file that contains information on the ethereum contract's public interfaces, such as events emitted and functions exposed by the contract. Both the deployment and invocation of contracts require Abi definitions.

CHAPTER 4
WEB-ENABLED LOAN SANCTION APPLICATION

Chapter 4

Web-Enabled Loan Sanction Application

4.1 Introduction

Blockchain is a decentralized ledger used to securely exchange digital currency and carry out deals and transactions. The banking systems can update from their traditional methodologies to a digital, immutable, distributed ledger that can be implemented via Blockchain. Many factors complicate Loan transactions' need for collateral, the time required for settlements, differences in currency denominations, third-party mediation, andmore. Multi-step processes, especially ones that require human interaction, are prime targets for fraudsters. With blockchain, information can be shared in real-time, and the ledger can only be updated when all parties agree. This can reduce time, costs, and opportunities to commit fraud. This project deals with the development of an online transaction application using Ganache and Metamask Wallet by the means of Ethereum as a digital currency.

4.2 MOTIVATION AND JUSTIFICATION

In Today's world, there are more fraudulent attacks on sanctioning loans day by day. Against these attacks the blockchain platforms establish a network and securely share details about transactions . This project is aimed to develop Web-enabled secured loan sanction application using smart contracts on blockchain technology.

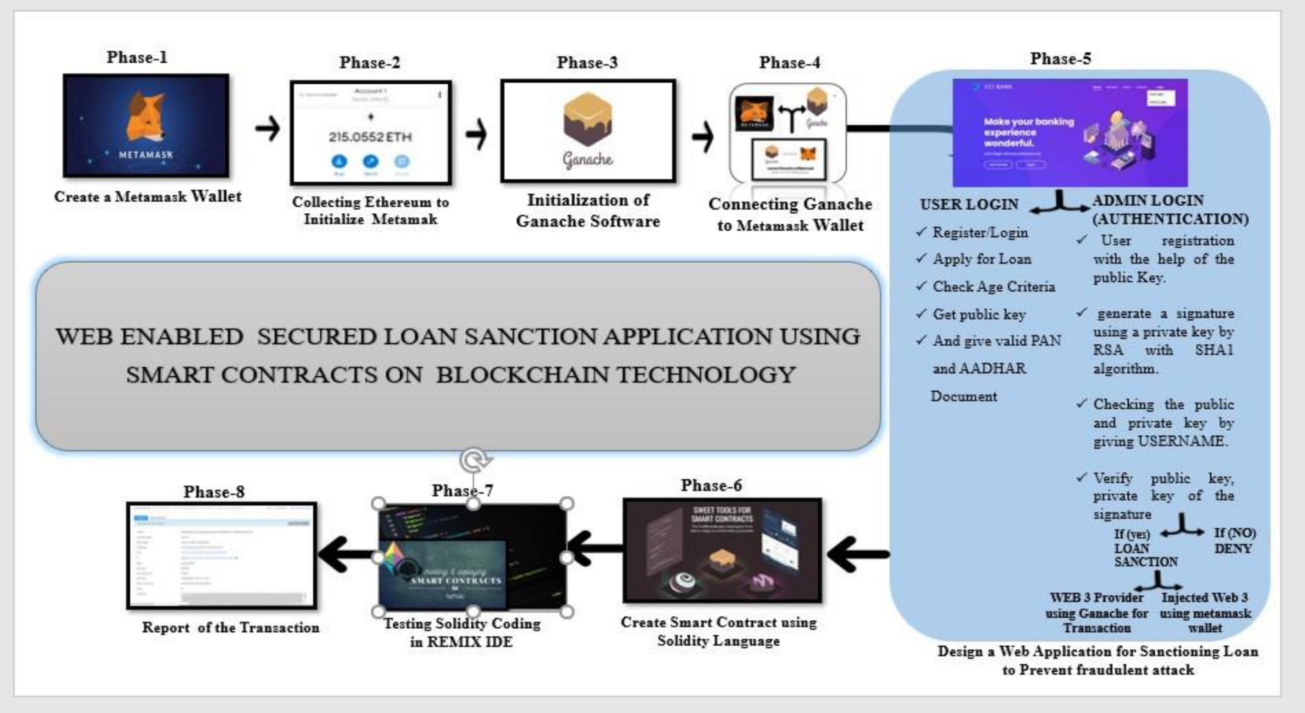
4.3 PROBLEM STATEMENT

- A Blockchain based platform mitigates fraud by establishing a network and securely sharing details about transactions between institutions in real-time.
- The system allows the institution to maintain the privacy of valuable customer data while automatically detecting any elements of fraud, small or large.

4.4 OBJECTIVE

The main purpose is to build the Ethereum blockchain which focuses on preventing such fraudulent attacks on sanctioning of Loans by decentralizing the processes. The security aspects concerning user identity authentication, bank official authentication, and multilevel verification of details are implemented by incorporating the notion of public key infrastructure (PKI).

4.5 Methodology



Phase-1
Create a Metamask Wallet
Phase-2
215.0552 ETH
Collecting Ethereum to Initialize Metamak
Phase-3
Ganache
Initialization of Ganache Software
Phase-4
Connecting Ganache to Metamask Wallet
Phase-5
Make your banking experience wonderful.
USER LOGIN
✓ Register/Login
✓ Apply for Loan
✓ Check Age Criteria
✓ Get public key
✓ And give valid PAN and AADHAR Document
ADMIN LOGIN (AUTHENTICATION)
✓ User registration with the help of the public Key.
✓ generate a signature using a private key by RSA with SHA1 algorithm.
✓ Checking the public and private key by giving USERNAME.
✓ Verify public key, private key of the signature
If (yes) LOAN SANCTION
If (NO) DENY
WEB 3 Provider using Ganache for Transaction
Injected Web 3 using metamask wallet
Design a Web Application for Sanctioning Loan to Prevent fraudulent attack
WEB ENABLED SECURED LOAN SANCTION APPLICATION USING SMART CONTRACTS ON BLOCKCHAIN TECHNOLOGY
Phase-6
Create Smart Contract using Solidity Language
Phase-7
Testing Solidity Coding in REMIX IDE
Phase-8
Report of the Transaction

Figure 4.1 Purposed Methodology

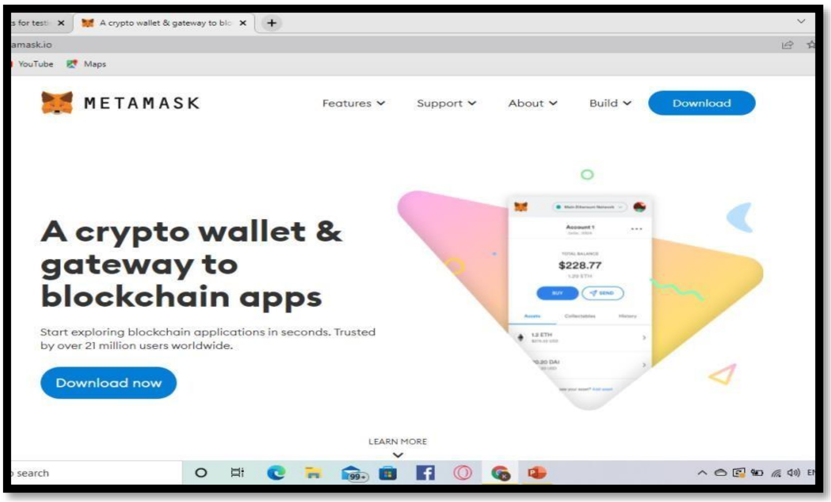
The entire methodology is divided into Eight phases namely, Creating a Metamask Wallet, Collecting Ethereum to Initialize Metamask, Initialization of Ganache Software, Collecting Ganache to Metamask Wallet, Designing a Web Application for sanctioning Loanto Prevent Fraudulent attack, Create Smart Contract using Solidity Language, Testing SolidityCoding in Remix IDE, Report of Transaction.

4.6 Implementation Approach

The Ethereum simulator Ganache is used in this work and the testnet Ropsten is used to build,implement, and test the DApp. In this project, both approaches are explained with DApp UI demo screenshots.

4.6.1 Phase 1: Create Metamask Wallet

- First, go to the Chrome Search and Search Metamask. Click on the MetaMask extension and install it.
- After the installation, go to the Chrome menu and click the **Add-ons** button. Then, click **Extensions** and then **MetaMask**.
- MetaMask will pop up and prompt user to sign in. If user doesn't have an account, user can create one by clicking the **Create account** button. After that, user can sign in to MetaMask.



A crypto wallet & gateway to blockchain apps

Start exploring blockchain applications in seconds. Trusted by over 21 million users worldwide.

Figure 4.2 Download Metamask

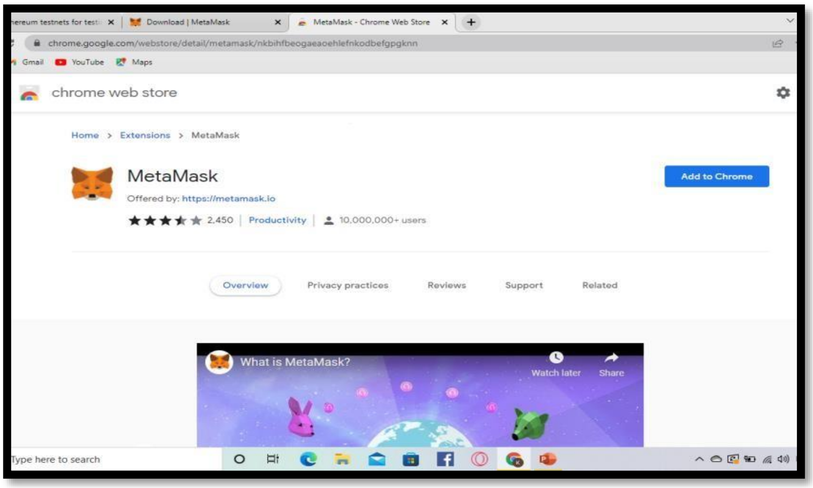
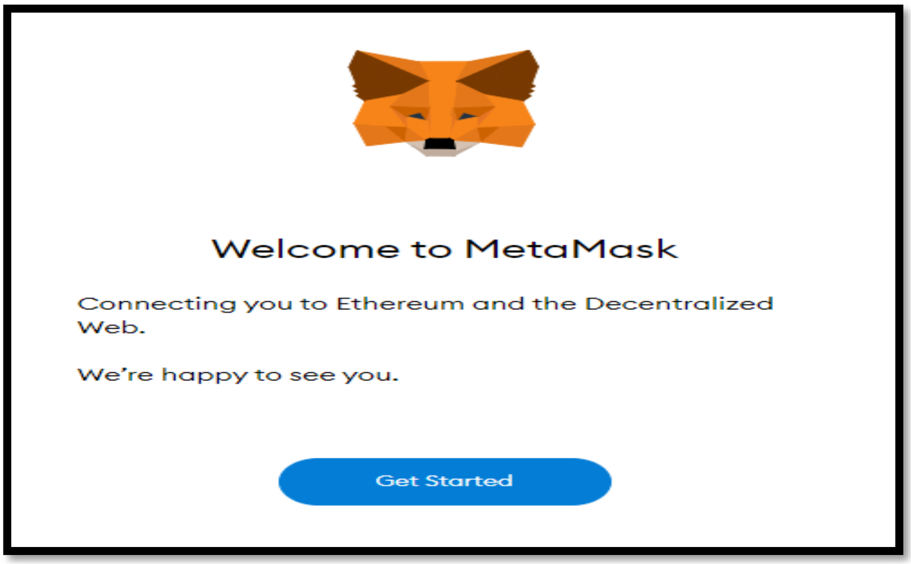

hereum testnets for test
Download | MetaMask
MetaMask - Chrome Web Store
chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn
Gmail
YouTube
Maps
chrome web store
Home > Extensions > MetaMask
MetaMask
Offered by: https://metamask.io
2,450 | Productivity | 10,000,000+ users
Add to Chrome
Overview
Privacy practices
Reviews
Support
Related
What is MetaMask?
Watch later
Share
Type here to search

Figure 4.3 Add Extension

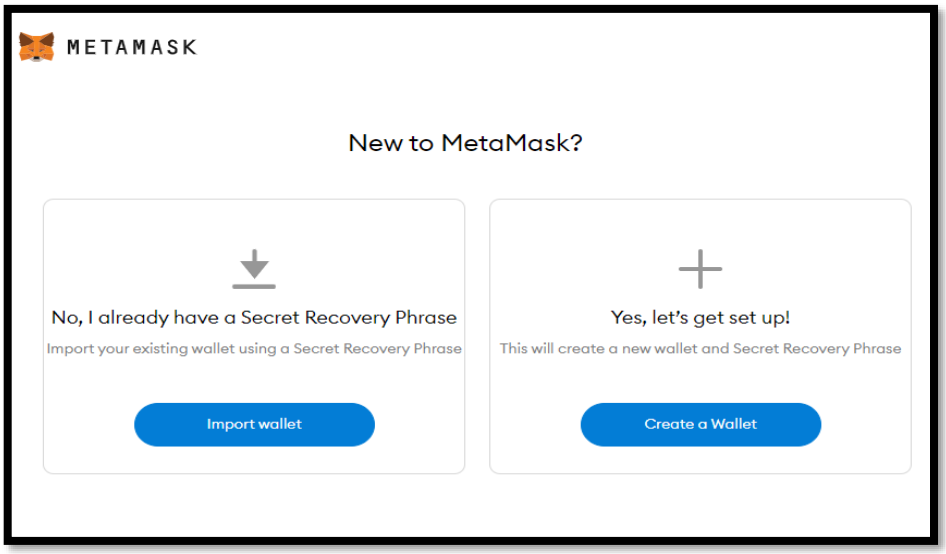


Welcome to MetaMask

Connecting you to Ethereum and the Decentralized Web.

We're happy to see you.

Figure 4.4 Get started



METAMASK

New to MetaMask?

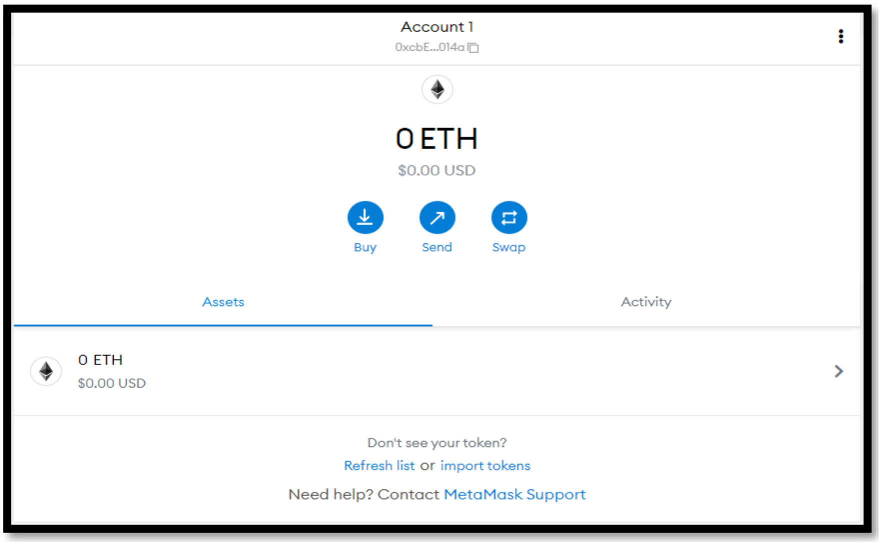
No, I already have a Secret Recovery Phrase

Import your existing wallet using a Secret Recovery Phrase

Yes, let's get set up!

This will create a new wallet and Secret Recovery Phrase

Figure 4.5 Create/Import Account



Account 1

0xcbE...014a

0 ETH

$0.00 USD

0 ETH

$0.00 USD

Don't see your token?

Refresh list or import tokens

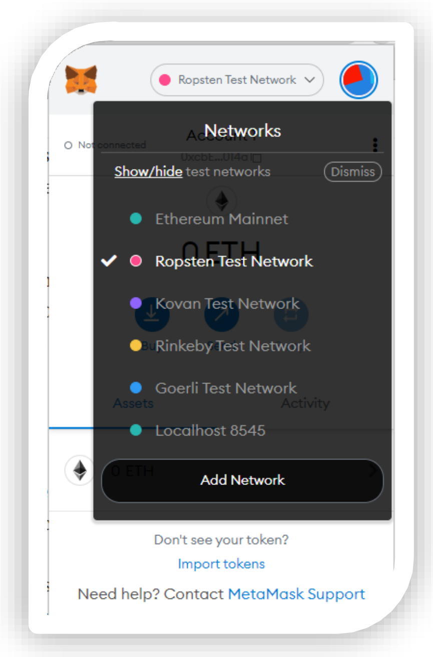
Need help? Contact MetaMask Support

Figure 4.6 Metamask Account

4.6.2 PHASE 2: Collecting Ethereum to Initialize Metamask

- Collecting a Ethereum to initialize Meta mask by Ethereum token
- user can easily buy Ethereum with a debit card or Apple Pay directly within MetaMask by clicking “Add funds”. user can request funds from a friend by sending them a payment request showing QR code in person or by sharing public address.

In this Metamask there are top 4 testnet for testing SMART CONTRACT



- Ropsten Test Network
- Kovan Test Network
- Rinkeby Test Network
- Goerli Test Network

Figure 4.7 Testnets

In this project, we used Ropstentestnet for collecting ETH

- The **ROPSTEN TESTNET** shows the latest blocks and latest transactions.
- To get free Eth on the Ropsten testnet, we can use the official faucet.
- Just paste Eth address and click **Send me test Ether** to get a free 0.3 Eth.

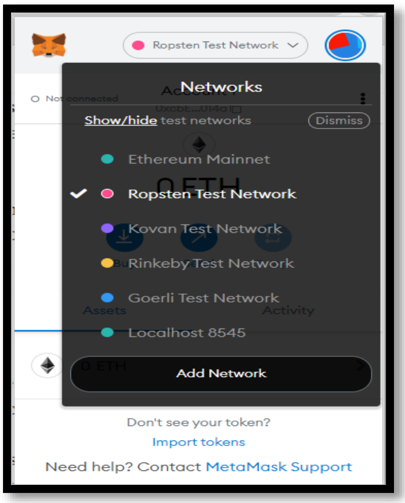


Figure 4.8 In Metamask click Ropsten testnet

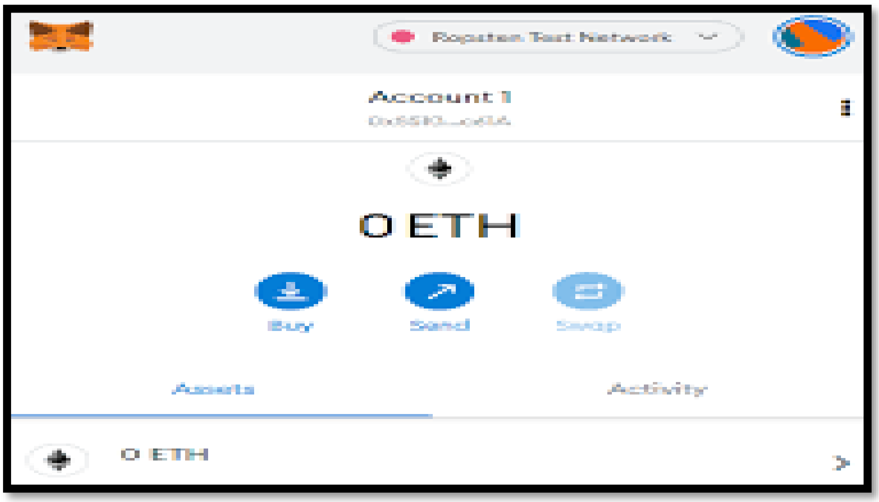
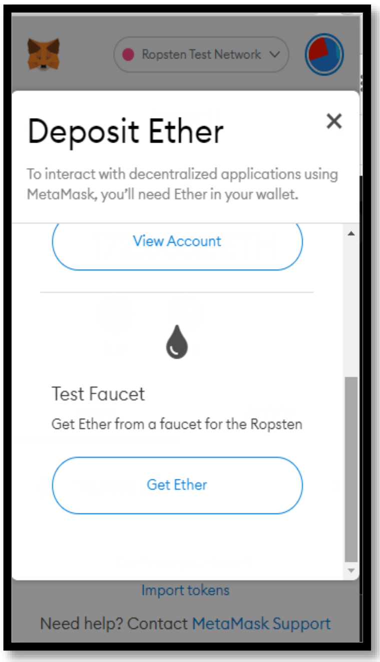


Figure 4.9 Click "BUY"



Deposit Ether

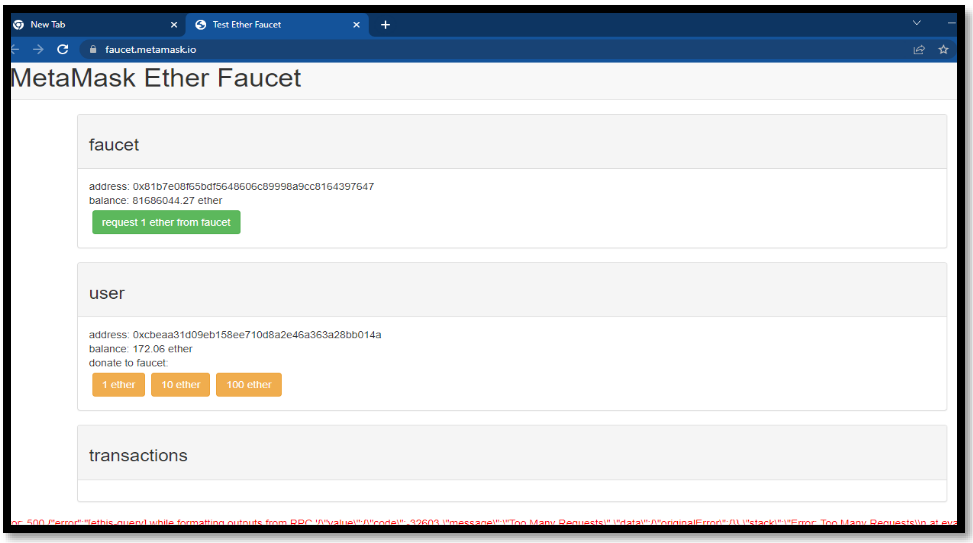
To interact with decentralized applications using MetaMask, you'll need Ether in your wallet.

Test Faucet

Get Ether from a faucet for the Ropsten

Need help? Contact MetaMask Support

Figure 4.10 Click "GET ETHER"



MetaMask Ether Faucet

faucet

address: 0x81b7e08f65bdf5648606c89998a9cc8164397647
balance: 81686044.27 ether

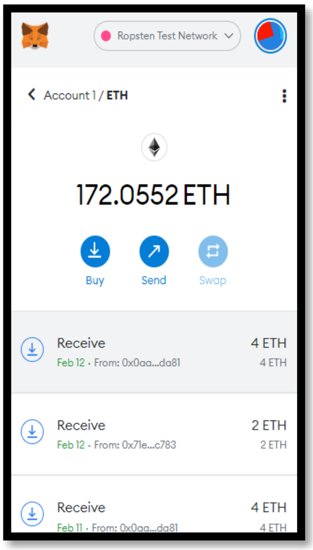
user

address: 0xcbeaa31d09eb158ee710d8a2e46a363a28bb014a
balance: 172.06 ether
donate to faucet:

transactions

```
or: 500 {"error":"[ethjs-query] while formatting outputs from RPC '{\"value\":{\"code\":-32603,\"message\":\"Too Many Requests\",\"data\":{\"originalError\":{}},\"stack\":\"Error: Too Many Requests\n at eva
```

Figure 4.11 Test Faucet



Account 1 / ETH

172.0552 ETH

Receive 4 ETH
Feb 12 · From: 0x0aa...da81 4 ETH

Receive 2 ETH
Feb 12 · From: 0x7le...c783 2 ETH

Receive 4 ETH
Feb 11 · From: 0x0aa...da81 4 ETH

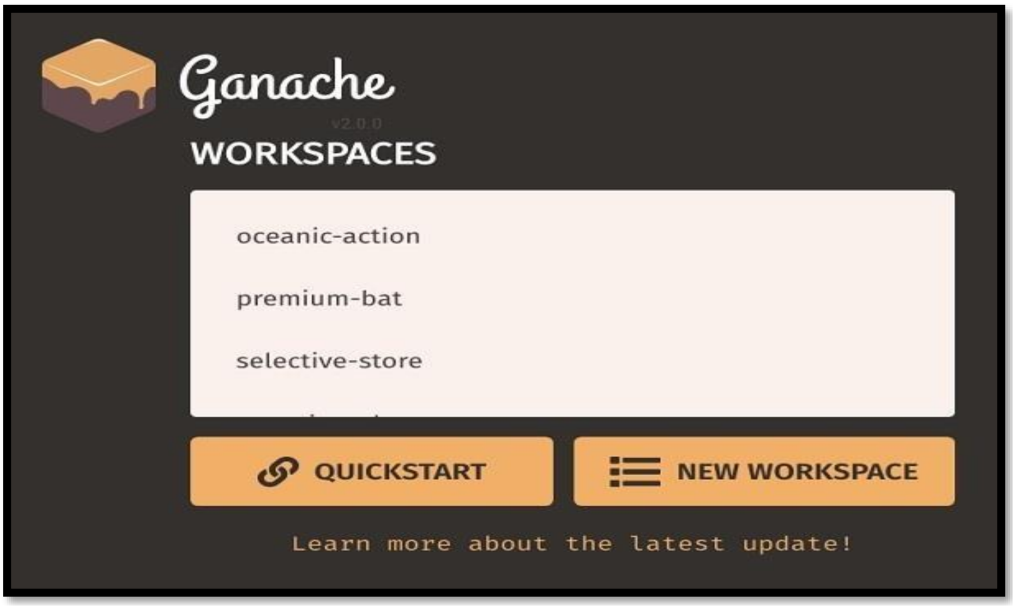
Figure 4.12 Collection of Ether

4.6.3 PHASE 3 : Initialization of GANACHE SOFTWARE

- Ganache is used for setting up a personal Ethereum Blockchain for testing Solidity contracts.
- Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development.
- user can use Ganache across the entire development cycle; enabling to develop, deploy, and test dapps(Decentralized Applications) in a safe and deterministic environment.

Quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.

Figure 4.13 Download Ganache

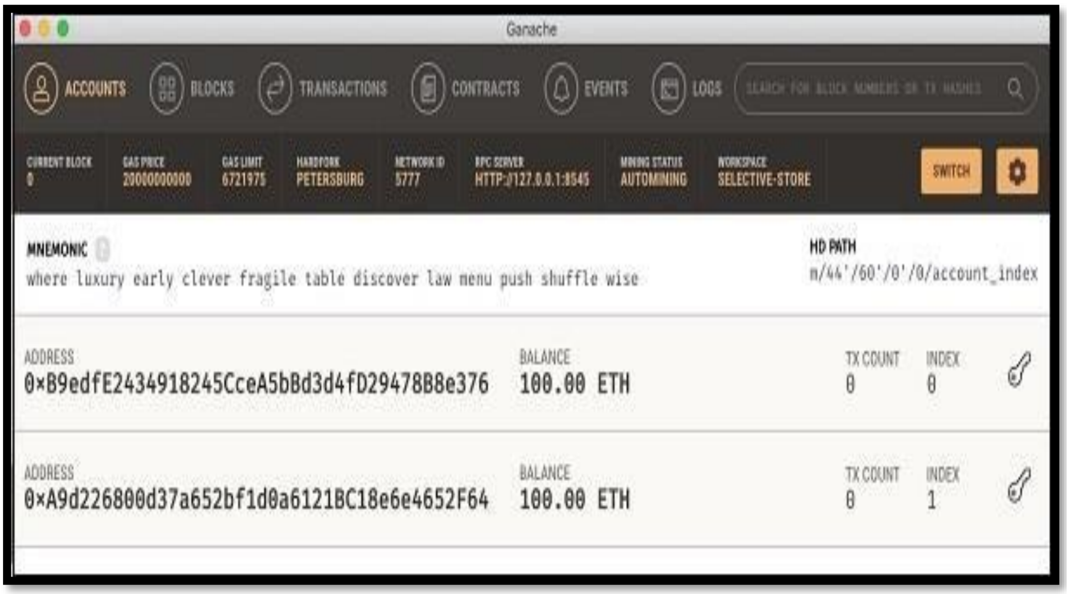


oceanic-action

premium-bat

selective-store

Figure 4.14 Workspaces



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
0	20000000000	6721975	PETERSBURG	5777	HTTP://127.0.0.1:8545	AUTOMINING	SELECTIVE-STORE

MNEMONIC
where luxury early clever fragile table discover law menu push shuffle wise

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×B9edfE2434918245CceA5bBd3d4fD29478B8e376	100.00 ETH	0	0
0×A9d226800d37a652bf1d0a6121BC18e6e4652F64	100.00 ETH	0	1

Figure 4.15 QuickStart

4.6.4 PHASE 4: Connecting Ganache to Metamask wallet

- ➢ Connecting Ganache to Metamask wallet is the Major role for making smart contract to deploy the contract and do transaction.
- ➢ import Ganache account to Metamask wallet and do transactions.

- Click on network and select custom RPC
- Give any name and provide the Ganache RPC Http URL.
- This will connect Metamask to Ganache. Initially balance will be 0 ether to import ganache account.
- Open accounts by clicking at top right corner of Metamask and select Import Account.
- Its necessary to provide private key of account in Ganache. Open Ganache and click on Show Keys for any of accounts which will show account address and private key. Copy the private key and paste it in metamask.

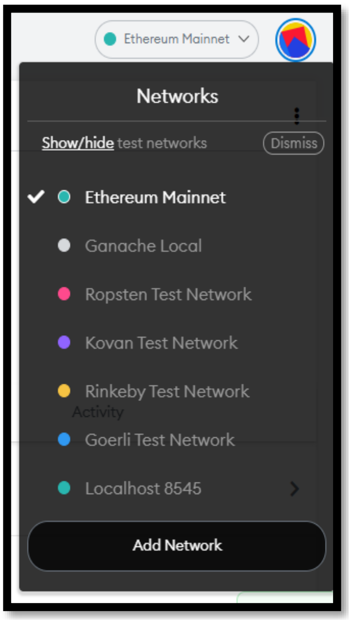
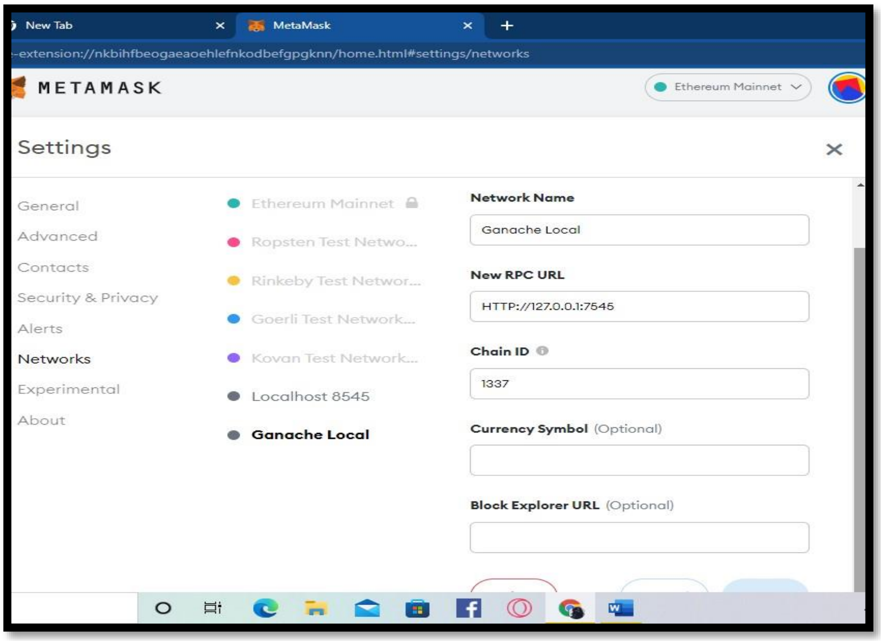


Figure 4.16 Add Network/Custom RPC


New Tab
MetaMask
-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#settings/networks
METAMASK
Ethereum Mainnet

Settings

- General
- Advanced
- Contacts
- Security & Privacy
- Alerts
- Networks
- Experimental
- About

- Ethereum Mainnet
- Ropsten Test Netwo...
- Rinkeby Test Networ...
- Goerli Test Network...
- Kovan Test Network...
- Localhost 8545
- **Ganache Local**

Network Name: Ganache Local

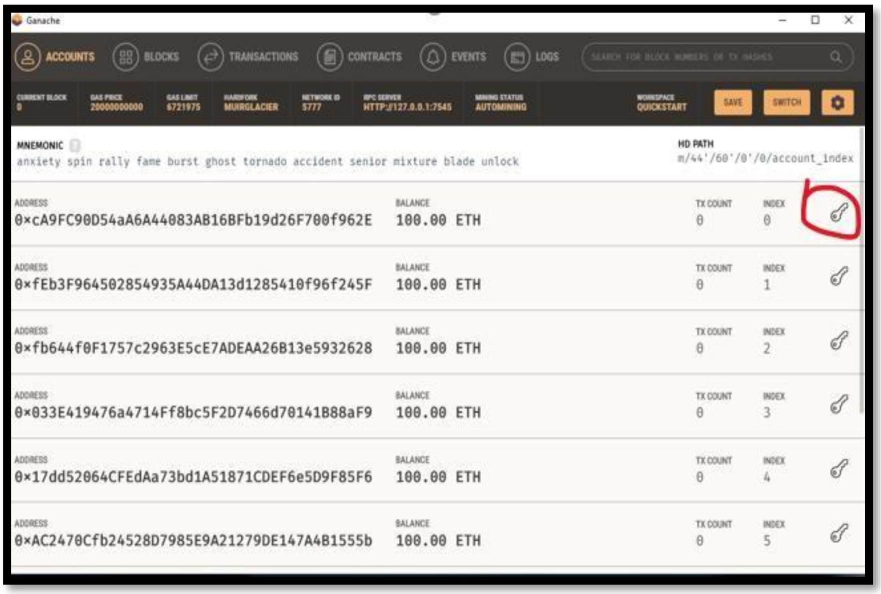
New RPC URL: HTTP://127.0.0.1:7545

Chain ID: 1337

Currency Symbol (Optional):

Block Explorer URL (Optional):

Figure 4.17 Provide URL



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
0	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:7545	AUTOMINING	QUICKSTART

MNEMONIC
anxiety spin rally fame burst ghost tornado accident senior mixture blade unlock

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×cA9FC90D54aA6A44083AB16BFb19d26F700f962E	100.00 ETH	0	0
0×fEb3F964502854935A44DA13d1285410f96f245F	100.00 ETH	0	1
0×fb644f0F1757c2963E5cE7ADEAA26B13e5932628	100.00 ETH	0	2
0×033E419476a4714Ff8bc5F2D7466d70141B88aF9	100.00 ETH	0	3
0×17dd52064CFEdAa73bd1A51871CDEF6e5D9F85F6	100.00 ETH	0	4
0×AC2470Cfb24528D7985E9A21279DE147A4B1555b	100.00 ETH	0	5

Figure 4.18 Copy private key

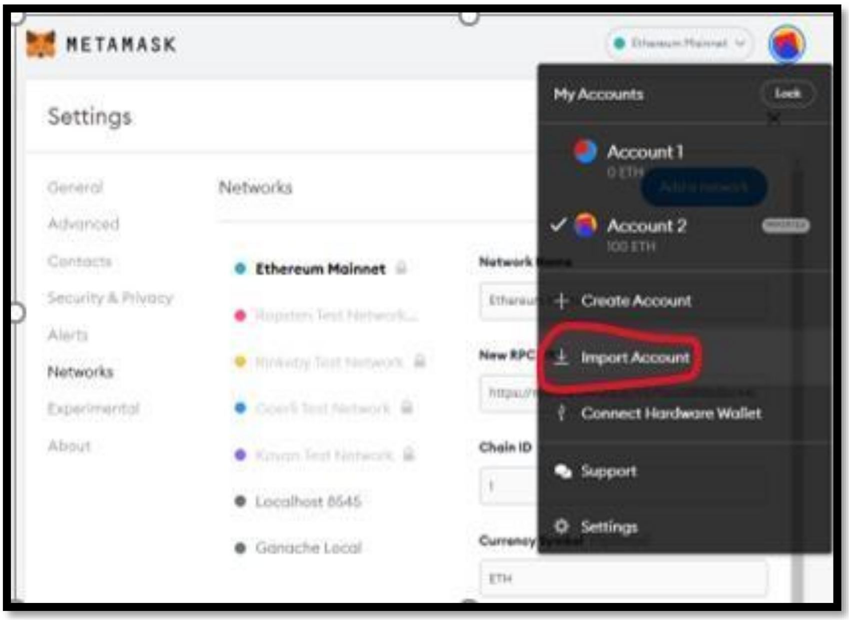
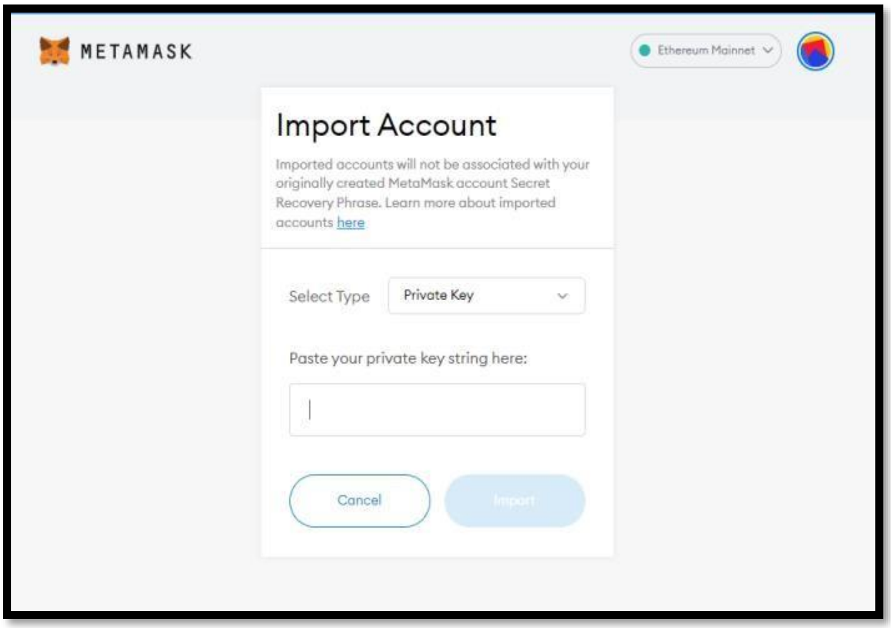


Figure 4.19 Import Account



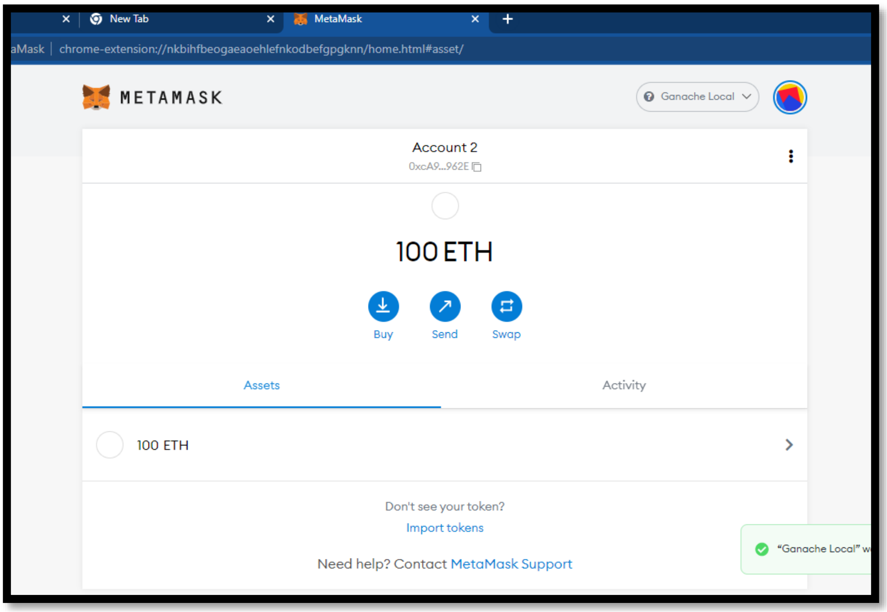
Import Account

Imported accounts will not be associated with your originally created MetaMask account Secret Recovery Phrase. Learn more about imported accounts here

Select Type: Private Key

Paste your private key string here:

Figure 4.20 Paste Private key



Don't see your token?

Need help? Contact MetaMask Support

Figure 4.21 Connected Ganache to metamask

4.6.5 PHASE 5: DESIGN A WEB APPLICATION FOR SANCTIONING LOAN

- Designed a Responsive web application using PHP,JS, MYSQL,HTML, CSS, BOOTSTRAP, AJAX, ANGULAR JS, and then start the Apache HTTP server and MySQL using XAMPP.

- In the application there are two logins
  - USER LOGIN
  - ADMIN LOGIN

- In USER LOGIN

Loan application form is there so that users can apply for a loan with the eligible Age Criteria and get a public key with the terms and conditions.

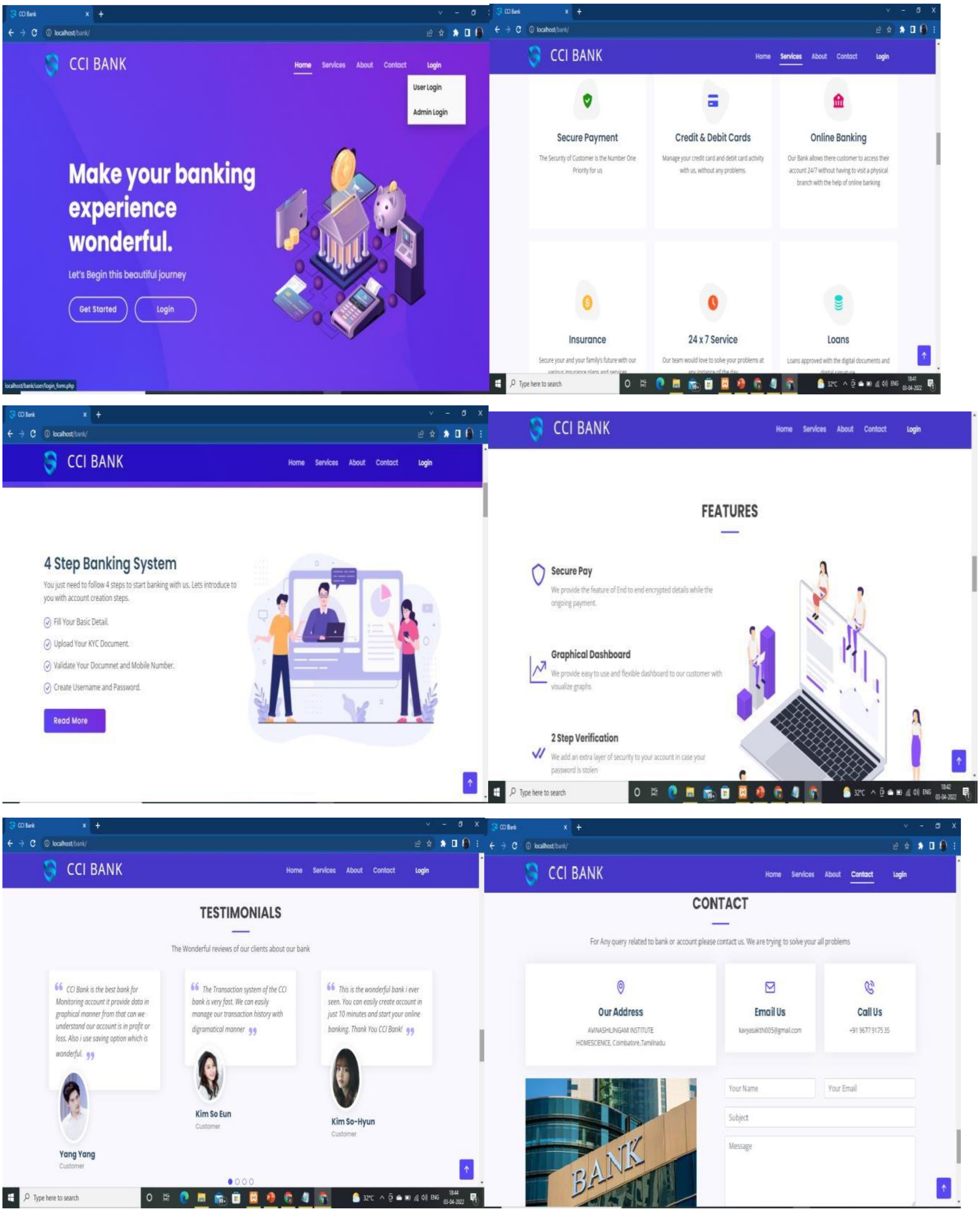
- In ADMIN LOGIN

- Generate Public key and private key of Customer signature using RSA(**Rivest–Shamir–Adleman) with sha 1 Algorithm.**

- The RSA algorithm is an **asymmetric cryptography algorithm**; this means that it uses a public key and a private key (i.e two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.

- SHA-1 or Secure Hash Algorithm 1 is **a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value**. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long.

- ➢ In admin login, Register with the public key and login with username and password.
- ➢ And then enter with private key to generate the signature and save sign so that signature will be stored in background MySQL and then log out.
- ➢ On the next page AUTHENTICATE Welcome page the user name is entered so that the "PUBLIC KEY OF USER and SIGNATURE OF THE USER" will be authenticated if the private key is wrong or someone is accessing with the wrong privatekey then it will be INVALID.
- ➢ Next, the authenticator enters the loan details and signing the details by clicking the button GENERATE SIGN. After generating a signature then click the SANCTION LOAN that button takes to a REMIX IDE.



Make your banking experience wonderful.

Let's Begin this beautiful journey

Secure Payment

The Security of Customer is the Number One Priority for us

Credit & Debit Cards

Manage your credit card and debit card activity with us, without any problems.

Online Banking

Our Bank allows there customer to access their account 24/7 without having to visit a physical branch with the help of online banking

Insurance

Secure your and your family's future with our various insurance plans and services

24 x 7 Service

Our team would love to solve your problems at any instance of the day

Loans

Loans approved with the digital documents and digital signature

4 Step Banking System

You just need to follow 4 steps to start banking with us. Lets introduce to you with account creation steps.

- Fill Your Basic Detail.
- Upload Your KYC Document.
- Validate Your Documnet and Mobile Number.
- Create Username and Password.

FEATURES

Secure Pay

We provide the feature of End to end encrypted details while the ongoing payment.

Graphical Dashboard

We provide easy to use and flexible dashboard to our customer with visualize graphs

2 Step Verification

We add an extra layer of security to your account in case your password is stolen

TESTIMONIALS

The Wonderful reviews of our clients about our bank

CCI Bank is the best bank for Monitoring account it provide data in graphical manner from that can we understand our account is in profit or loss. Also i use saving option which is wonderful.

Yang Yang
Customer

The Transaction system of the CCI bank is very fast. We can easily manage our transaction history with digramatical manner

Kim So Eun
Customer

This is the wonderful bank i ever seen. You can easily create account in just 10 minutes and start your online banking. Thank You CCI Bank!

Kim So-Hyun
Customer

CONTACT

For Any query related to bank or account please contact us. We are trying to solve your all problems

Our Address

AVINASHILINGAM INSTITUTE
HOMESCIENCE, Coimbatore.,Tamilnadu

Email Us

kavyasakthi005@gmail.com

Call Us

+91 9677 9175 35

Figure 4.22 Designing Webpage

User Login

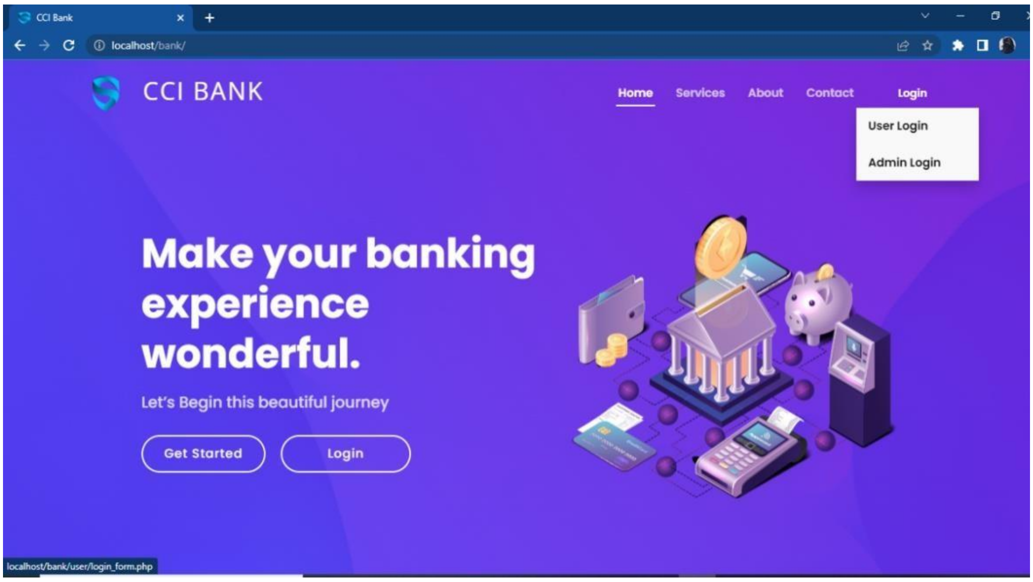

CCI Bank
localhost/bank/
CCI BANK
Home
Services
About
Contact
Login
User Login
Admin Login
Make your banking experience wonderful.
Let's Begin this beautiful journey
Get Started
Login
localhost/bank/user/login_form.php

Figure 4.23 Click user login

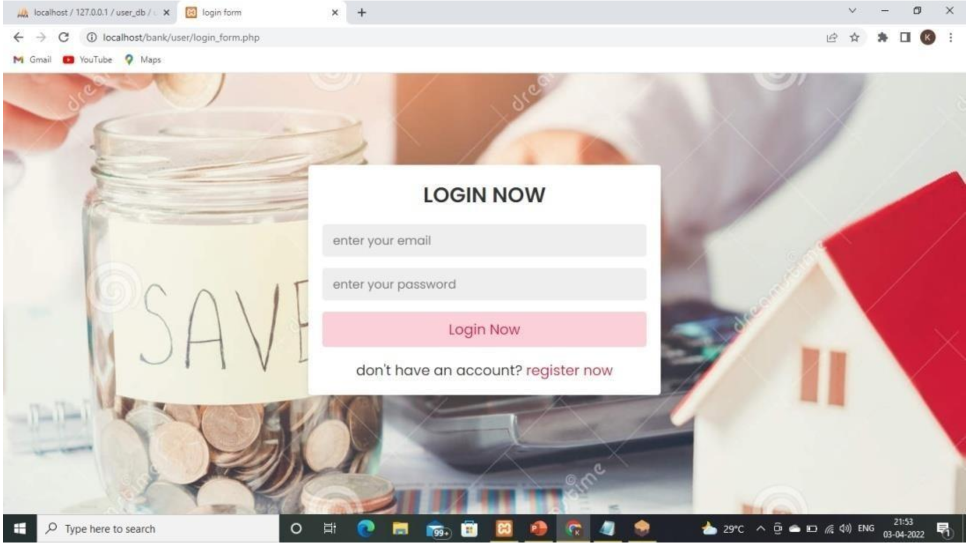


Figure 4.24 Register/login page

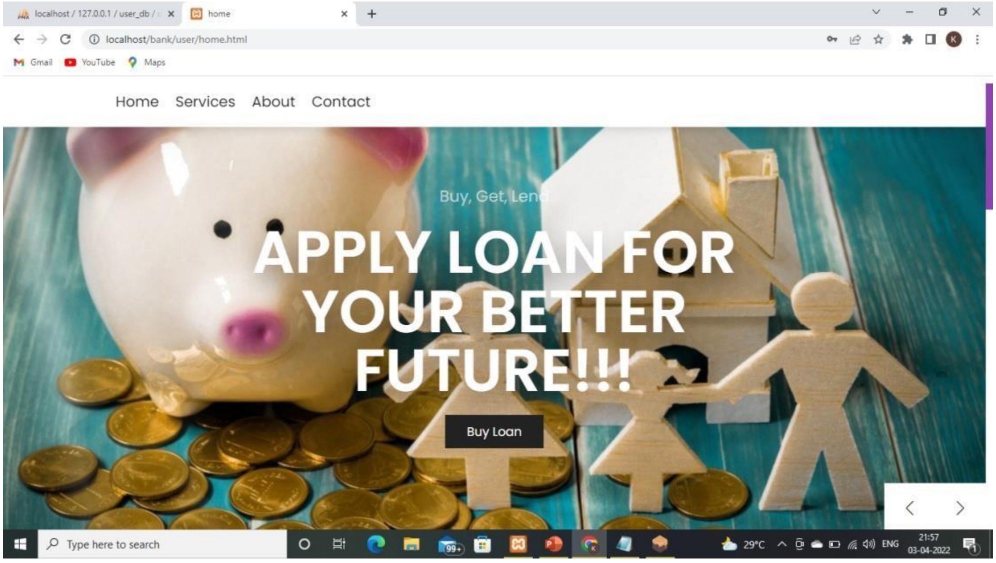
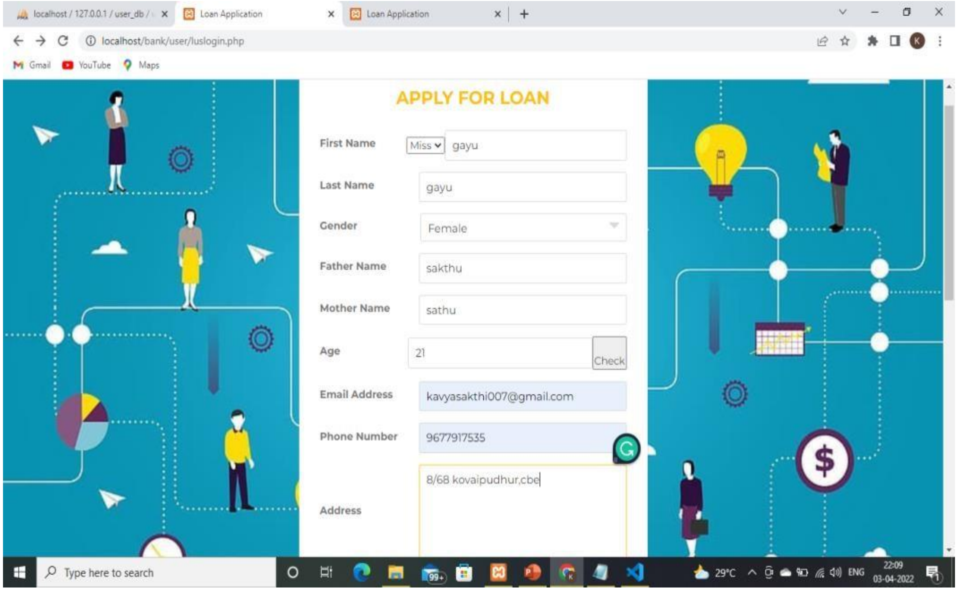


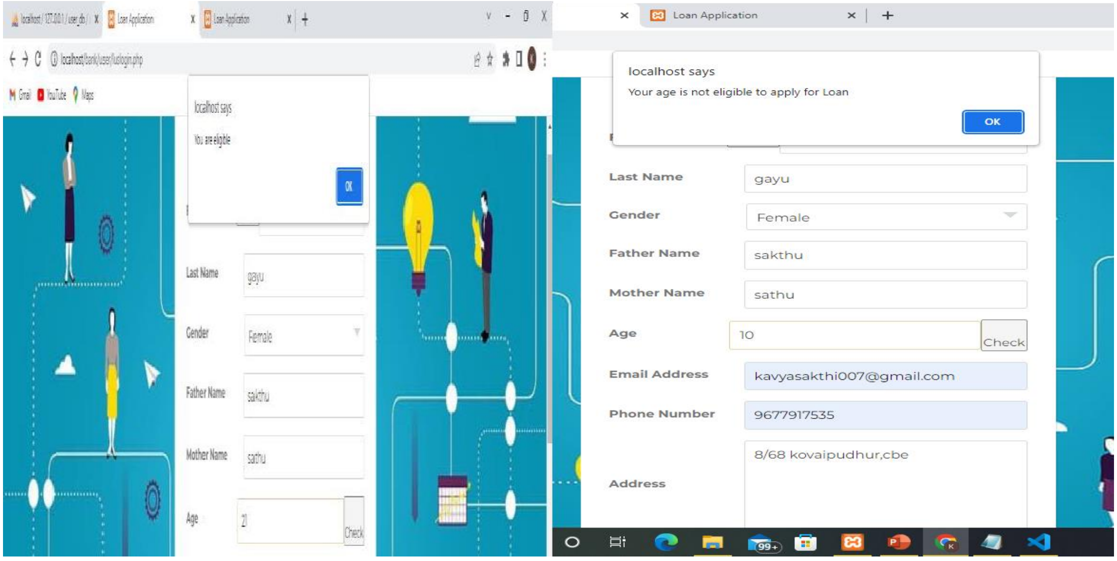
Figure 4.25 Click buy loan



APPLY FOR LOAN

First Name	Miss gayu
Last Name	gayu
Gender	Female
Father Name	sakthu
Mother Name	sathu
Age	21 Check
Email Address	kavyasakthi007@gmail.com
Phone Number	9677917535
Address	8/68 kovaipudhur,cbe

Figure 4.26 Apply for Loan



localhost says

You are eligible

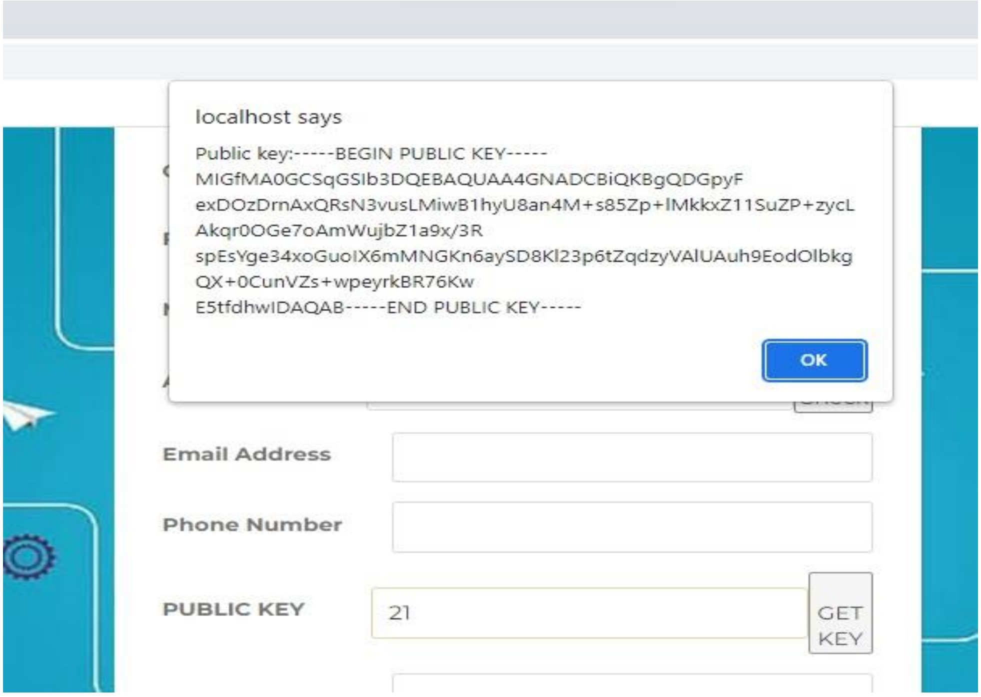
Last Name	gayu
Gender	Female
Father Name	sakthu
Mother Name	sathu
Age	21

localhost says

Your age is not eligible to apply for Loan

Last Name	gayu
Gender	Female
Father Name	sakthu
Mother Name	sathu
Age	10
Email Address	kavyasakthi007@gmail.com
Phone Number	9677917535
Address	8/68 kovaipudhur,cbe

Figure 4.27 Age criteria alert for applying loan

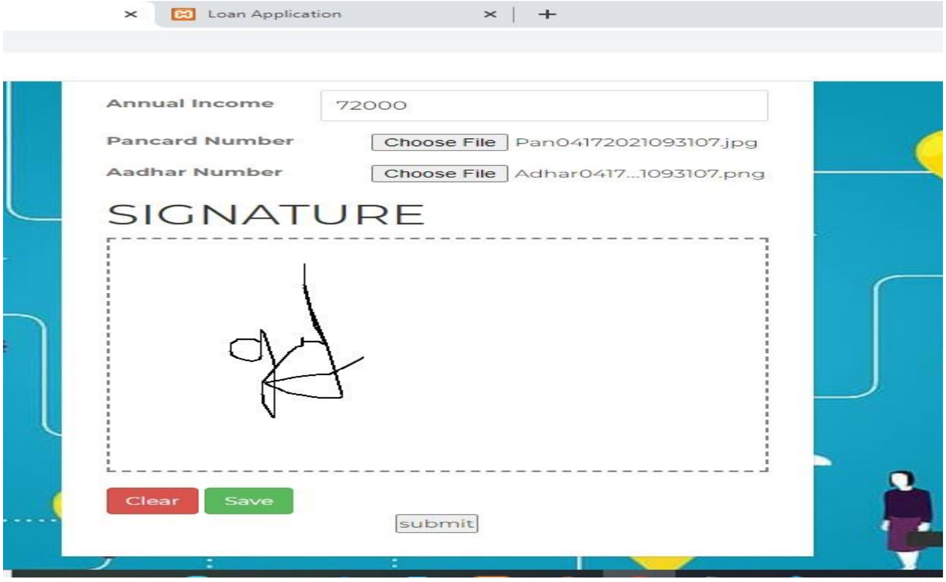


localhost says

```
Public key:-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyF
exDOzDrnAxQRsN3vusLMiwB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycL
Akqr0OGe7oAmWujbZ1a9x/3R
spEsYge34xoGuoIX6mMNGKn6aySD8Kl23p6tZqdzyVAlUAuh9EodOlbkg
QX+0CunVZs+wpeyrkBR76Kw
E5tfdhwIDAQAB-----END PUBLIC KEY-----
```

Email Address	
Phone Number	
PUBLIC KEY	21

Figure 4.28 "Get Public Key" by giving age



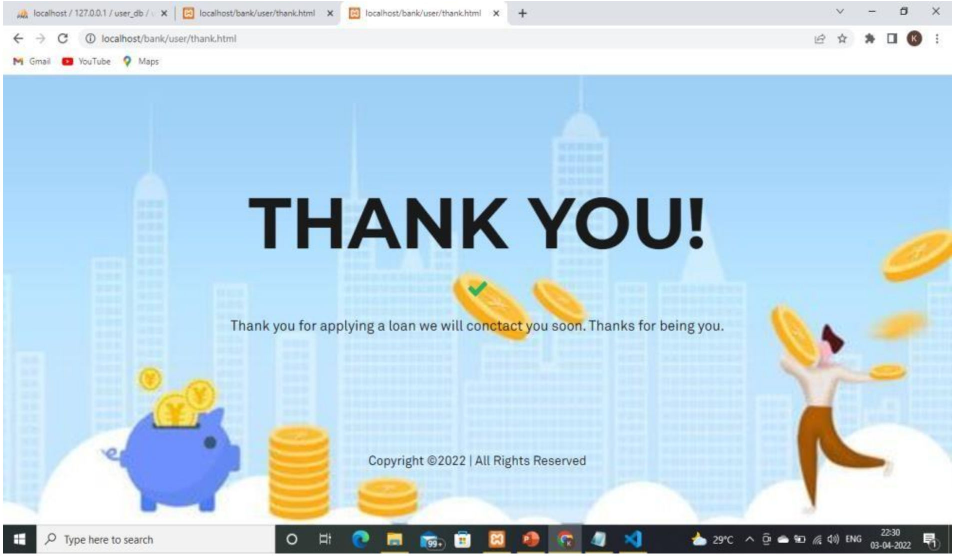
Annual Income 72000

Pancard Number Pan04172021093107.jpg

Aadhar Number Adhar0417...1093107.png

SIGNATURE

Figure 4.29 Applying loan


localhost / 127.0.0.1 / user_db /
localhost/bank/user/thank.html
localhost/bank/user/thank.html
Gmail
YouTube
Maps
Type here to search
29°C
ENG
22:30
03-04-2022

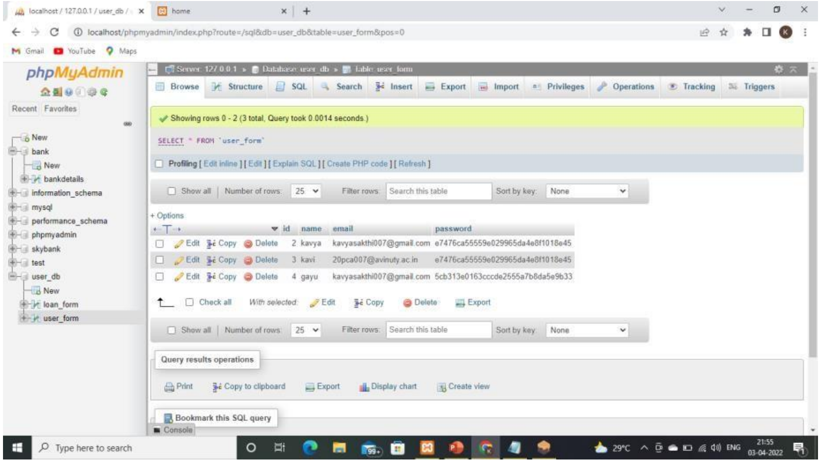
localhost/bank/user/thank.html

THANK YOU!

Thank you for applying a loan we will conctact you soon. Thanks for being you.

Copyright ©2022 | All Rights Reserved

Figure 4.30 Loan applied successfully



localhost/phpmyadmin/index.php?route=/sql&db=user_db&table=user_form&pos=0
phpMyAdmin
Server: 127.0.0.1 » Database: user_db » Table: user_form
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Triggers

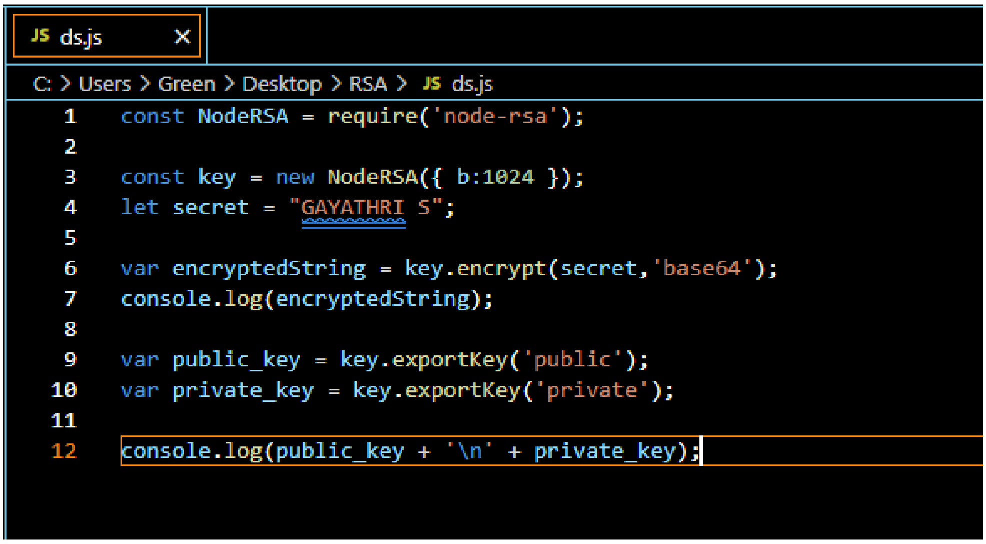
✓ Showing rows 0 - 2 (3 total, Query took 0.0014 seconds.)

```
SELECT * FROM `user_form`
```

id	name	email	password
2	kavya	kavyasakthi007@gmail.com	e7476ca55559e029965da4e8f1018e45
3	kavi	20pca007@avinuty.ac.in	e7476ca55559e029965da4e8f1018e45
4	gayu	kavyasakthi007@gmail.com	5cb313e0163cccde2555a7b8da5e9b33

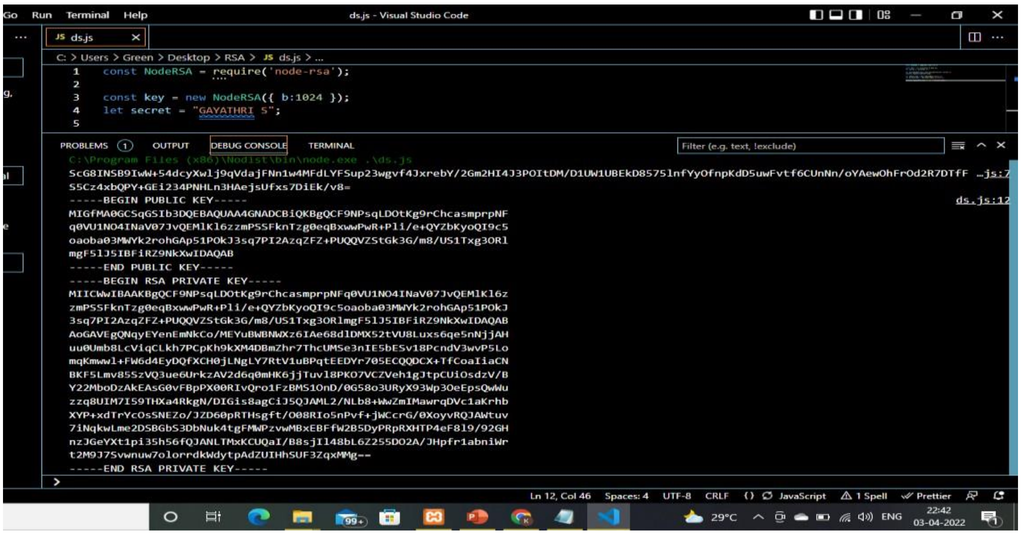
Figure 4.31 Data stored in MYSQL

Generating Public Key/Private Key in NODE.JS



```javascript
const NodeRSA = require('node-rsa');

const key = new NodeRSA({ b:1024 });
let secret = "GAYATHRI S";

var encryptedString = key.encrypt(secret,'base64');
console.log(encryptedString);

var public_key = key.exportKey('public');
var private_key = key.exportKey('private');

console.log(public_key + '\n' + private_key);
```

Figure 4.32 Code to generate a public/private of signature using node.js

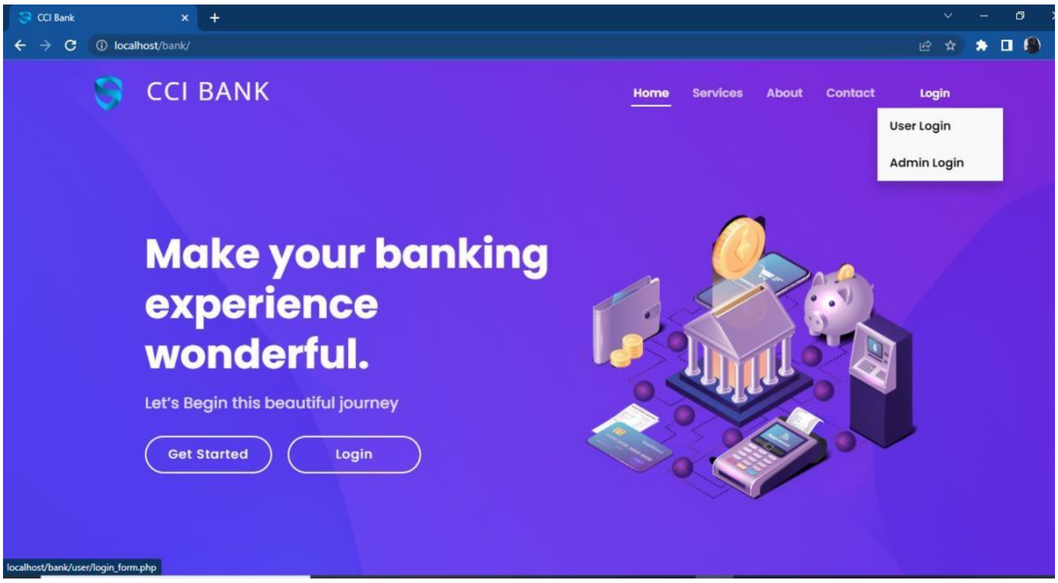


```
const NodeRSA = require('node-rsa');

const key = new NodeRSA({ b:1024 });
let secret = "GAYATHRI S";

```

```
C:\Program Files (x86)\Nodist\bin\node.exe .\ds.js
ScG8INSB9IwW+54dcyXwlj9qVdajFNn1w4MFdLYFSup23wgvf4JxrebY/2Gm2HI4J3POItDM/D1UW1UBEkD8575lnfYyOfnpKdD5uwFvtf6CUnNn/oYAewOhFrOd2R7DTfF
S5Cz4xbQPY+GEi234PNHLn3HAejsUfxs7DiEk/v8=
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCF9NPsqLDOtKg9rChcasmprpNF
q0VU1NO4INaV07JvQEMlKl6zzmPSSFknTzg0eqBxwwPwR+Pli/e+QYZbKyoQI9c5
oaoba03MWYk2rohGAp51POkJ3sq7PI2AzqZFZ+PUQQVZStGk3G/m8/US1Txg3ORl
mgF5lJ5IBFiRZ9NkXwIDAQAB
-----END PUBLIC KEY-----
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQCF9NPsqLDOtKg9rChcasmprpNFq0VU1NO4INaV07JvQEMlKl6z
zmPSSFknTzg0eqBxwwPwR+Pli/e+QYZbKyoQI9c5oaoba03MWYk2rohGAp51POkJ
3sq7PI2AzqZFZ+PUQQVZStGk3G/m8/US1Txg3ORlmgF5lJ5IBFiRZ9NkXwIDAQAB
AoGAVEgQNqyEYenEmNkCo/MEYuBWBNWXz6IAe68dlDMX52tVU8Luxs6qe5nNjjAH
uu0Umb8LcViqCLkh7PCpKh9kXM4DBmZhr7ThcUMSe3nIE5bESv18PcndV3wvP5Lo
mqKmwwl+FW6d4EyDQfXCH0jLNgLY7RtV1uBPqtEEDYr705ECQQDCX+TfCoaIiaCN
BKF5Lmv85SzVQ3ue6UrkzAV2d6q0mHK6jjTuvl8PKO7VCZVeh1gJtpCUiOsdzV/B
Y22MboDzAkEAsG0vFBpPX00RIvQro1FzBMS1OnD/0G58o3URyX93Wp3OeEpsQwWu
zzq8UIM7I59THXa4RkgN/DIGis8agCiJ5QJAML2/NLb8+WwZmIMawrqDVc1aKrhb
XYP+xdTrYcOsSNEZo/JZD60pRTHsgft/O08RIo5nPvf+jWCcrG/0XoyvRQJAWtuv
7iNqkwLme2DSBGbS3DbNuk4tgFMWPzvwMBxEBFfW2B5DyPRpRXHTP4eF8l9/92GH
nzJGeYXt1pi35h56fQJANLTMxKCUQaI/B8sjIl48bL6Z255DO2A/JHpfr1abniWr
t2M9J7Svwnuw7olorrdkWdytpAdZUIHhSUF3ZqxMMg==
-----END RSA PRIVATE KEY-----
```

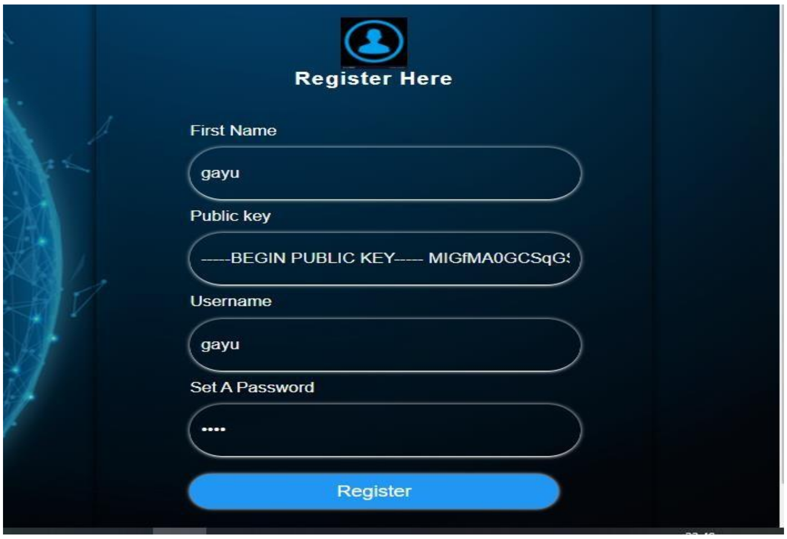
Figure 4.33 RSA Algorithm used to generate public/private key(SHA 1) Algorithm

ADMIN LOGIN



CCI Bank
localhost/bank/
CCI BANK
Home
Services
About
Contact
Login
User Login
Admin Login
Make your banking experience wonderful.
Let's Begin this beautiful journey
Get Started
Login
localhost/bank/user/login_form.php

Figure 4.34 Click "Admin Login"



Register Here

First Name

gayu

Public key

-----BEGIN PUBLIC KEY----- MIGfMA0GCSqG

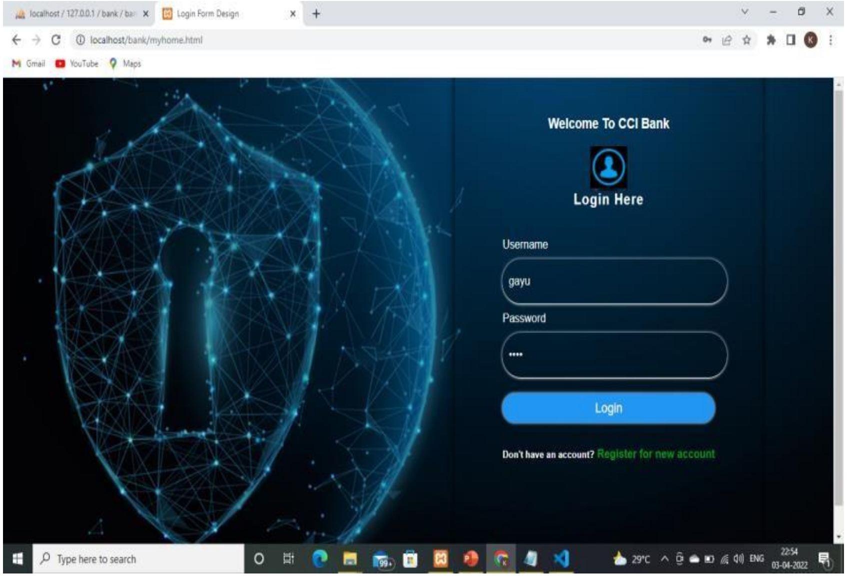
Username

gayu

Set A Password

••••

Figure 4.35 Register with public key



localhost / 127.0.0.1 / bank / ba
Login Form Design
localhost/bank/myhome.html
Gmail
YouTube
Maps

Welcome To CCI Bank

Login Here

Username

gayu

Password

••••

Login
Don't have an account? Register for new account
Type here to search
29°C
ENG
22:54
03-04-2022

Figure 4.36 Login page

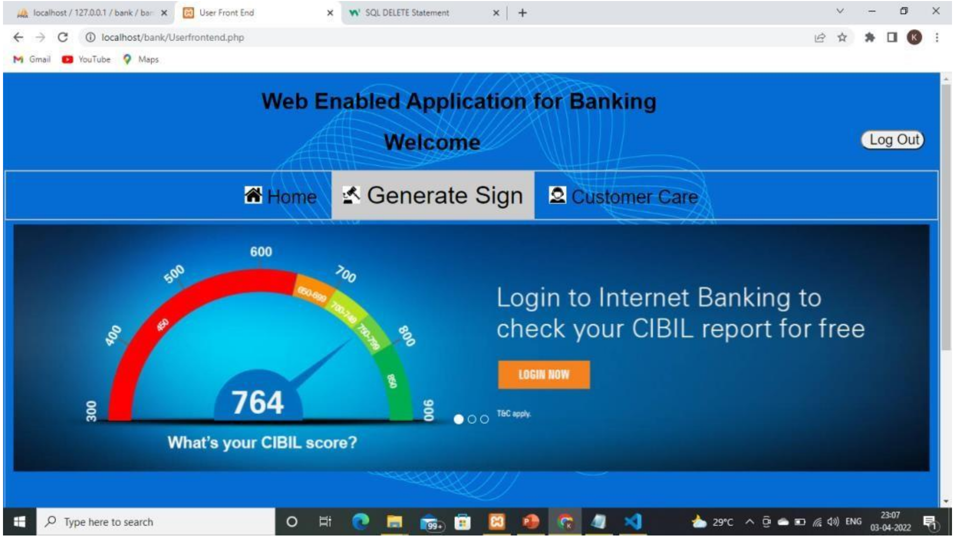
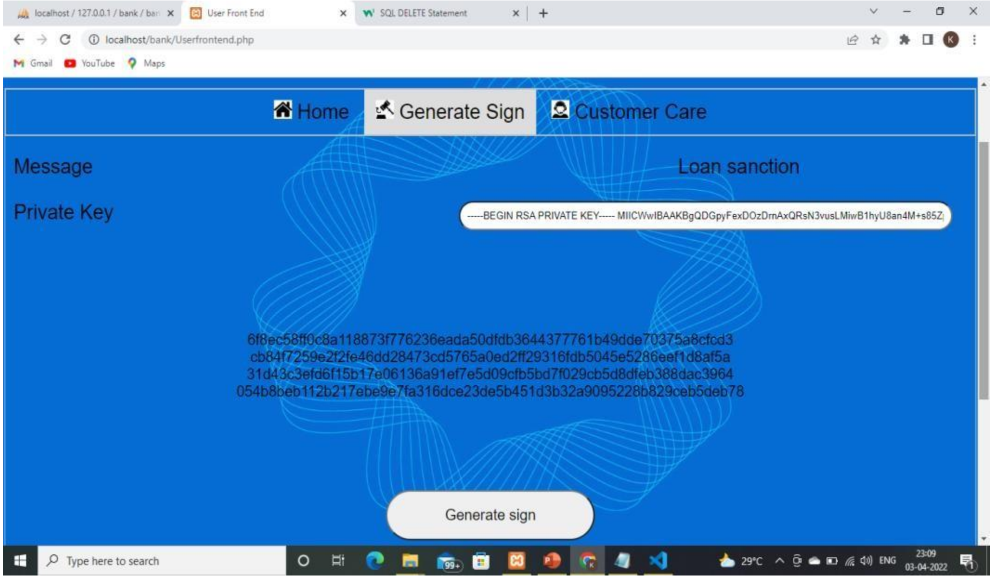

localhost/bank/Userfrontend.php
Web Enabled Application for Banking
Welcome
Log Out
Home
Generate Sign
Customer Care
Login to Internet Banking to check your CIBIL report for free
LOGIN NOW
What's your CIBIL score?
764

Figure 4.37 User frontend page to generate signature

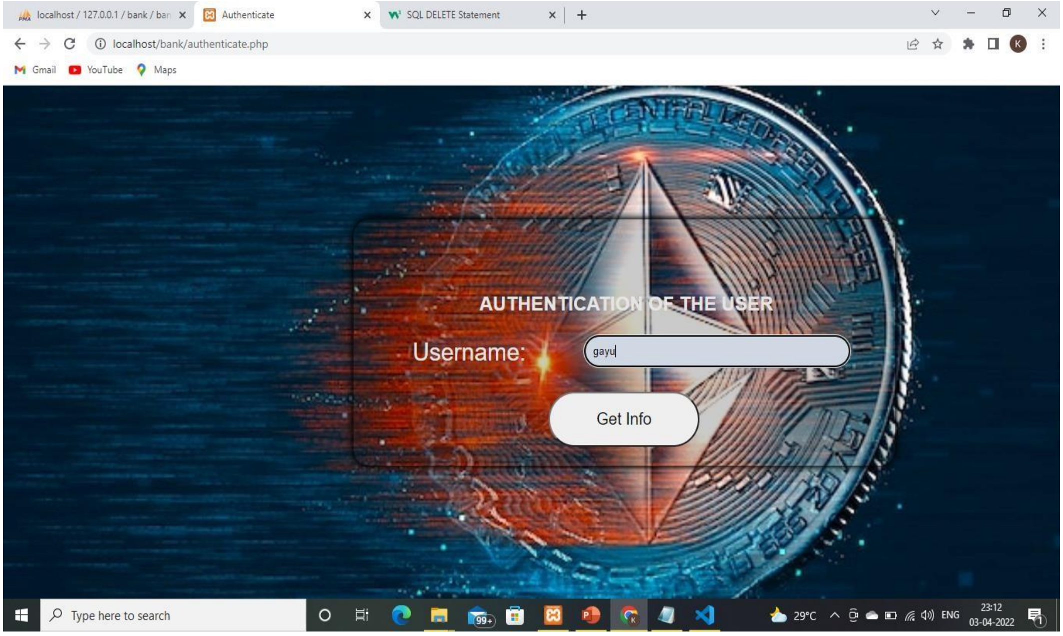


Message Loan sanction

Private Key -----BEGIN RSA PRIVATE KEY----- MIICWwIBAAKBgQDGpyFexDOzDmAxQRsN3vusLMiwB1hyU8an4M+s85Z

```
6f8ec58ff0c8a118873f776236eada50dfdb3644377761b49dde70375a8cfcd3
cb84f7259e2f2fe46dd28473cd5765a0ed2ff29316fdb5045e5286eef1d8af5a
31d43c3efd6f15b17e06136a91ef7e5d09cfb5bd7f029cb5d8dfeb388dac3964
054b8beb112b217ebe9e7fa316dce23de5b451d3b32a9095228b829ceb5deb78
```

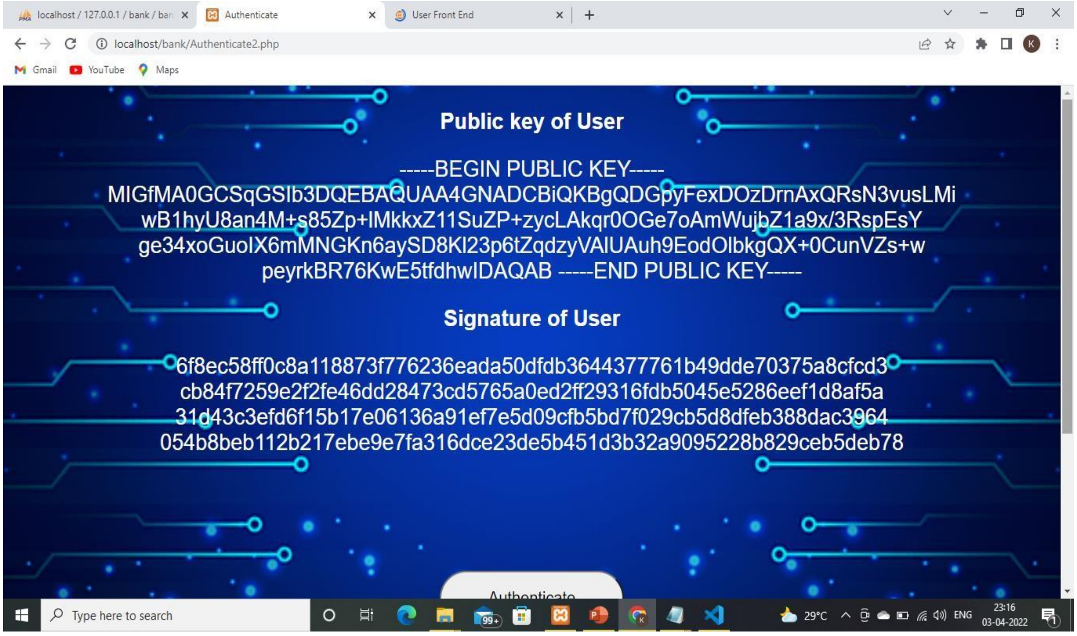
Figure 4.38 Generate a signature by entering private key and save



AUTHENTICATION OF THE USER

Username: gayu

Figure 4.39 Authentication of the user



Public key of User

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDrnAxQRsN3vusLMi
wB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycLAkqr0OGe7oAmWujbZ1a9x/3RspEsY
ge34xoGuoIX6mMNGKn6aySD8KI23p6tZqdzyVAIUAuh9EodOlbkgQX+0CunVZs+w
peyrkBR76KwE5tfdhwIDAQAB -----END PUBLIC KEY-----
```

Signature of User

```
6f8ec58ff0c8a118873f776236eada50dfdb3644377761b49dde70375a8cfcd3
cb84f7259e2f2fe46dd28473cd5765a0ed2ff29316fdb5045e5286eef1d8af5a
31d43c3efd6f15b17e06136a91ef7e5d09cfb5bd7f029cb5d8dfeb388dac3964
054b8beb112b217ebe9e7fa316dce23de5b451d3b32a9095228b829ceb5deb78
```

Figure 4.40 Public key &signature of the user authenticated successfully

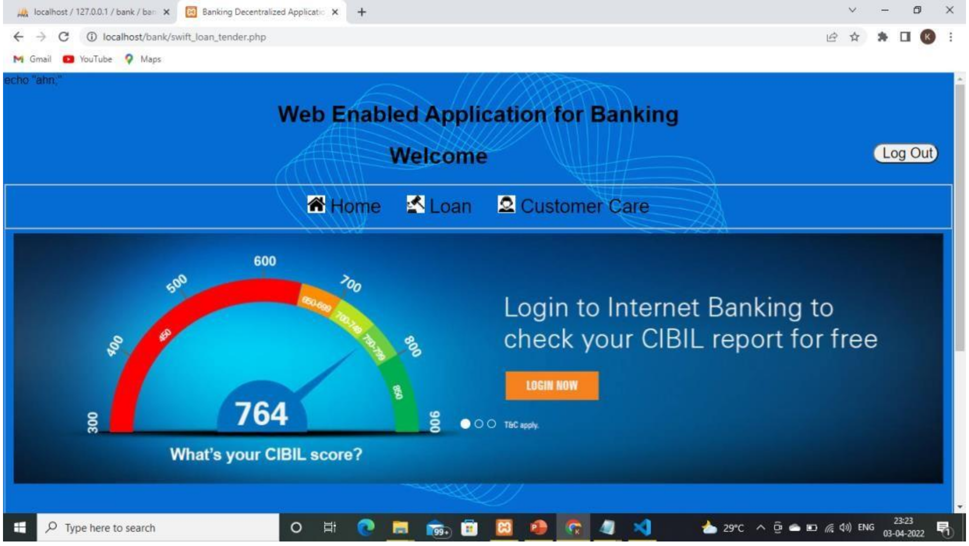
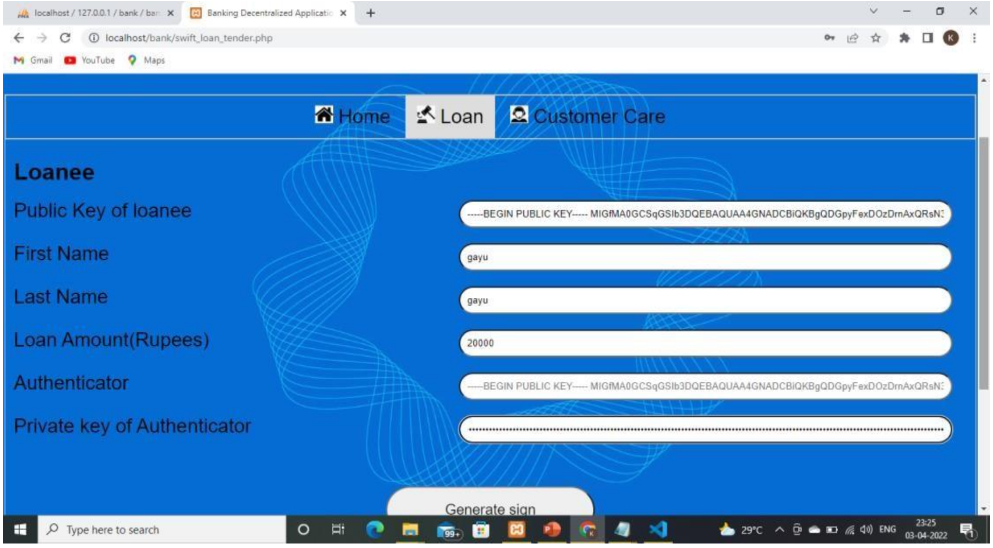

localhost / 127.0.0.1 / bank / ban
Banking Decentralized Applicatio
localhost/bank/swift_loan_tender.php
Gmail
YouTube
Maps
echo "ahri;"
Web Enabled Application for Banking
Welcome
Log Out
Home
Loan
Customer Care
300
400
500
600
700
800
900
764
What's your CIBIL score?
Login to Internet Banking to check your CIBIL report for free
LOGIN NOW
T&C apply.
Type here to search
29°C
ENG
23:23
03-04-2022

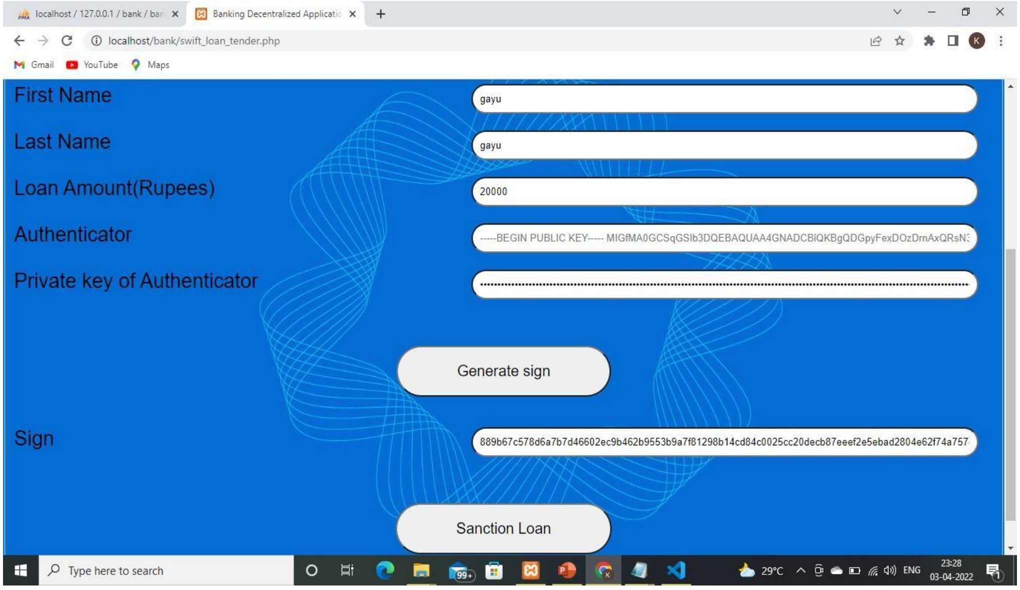
Figure 4.41 Loan page


localhost / 127.0.0.1 / bank / ban
Banking Decentralized Applicatio
localhost/bank/swift_loan_tender.php
Gmail
YouTube
Maps
Home
Loan
Customer Care
Generate sign
Type here to search
29°C
ENG
23:25
03-04-2022

Loanee

Public Key of loanee	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDmAxQRsN
First Name	gayu
Last Name	gayu
Loan Amount(Rupees)	20000
Authenticator	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDmAxQRsN
Private key of Authenticator	••

Figure 4.42 Give loan details and give private key


localhost / 127.0.0.1 / bank / ban
Banking Decentralized Applicatio
localhost/bank/swift_loan_tender.php
Gmail
YouTube
Maps
Type here to search
29°C
ENG
23:28
03-04-2022

First Name gayu

Last Name gayu

Loan Amount(Rupees) 20000

Authenticator -----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDrnAxQRsN3

Private key of Authenticator ••

Generate sign

Sign 889b67c578d6a7b7d46602ec9b462b9553b9a7f81298b14cd84c0025cc20decb87eeef2e5ebad2804e62f74a757

Sanction Loan

Figure 4.43 Click "generate sign" and signature generates and click "sanction loan"

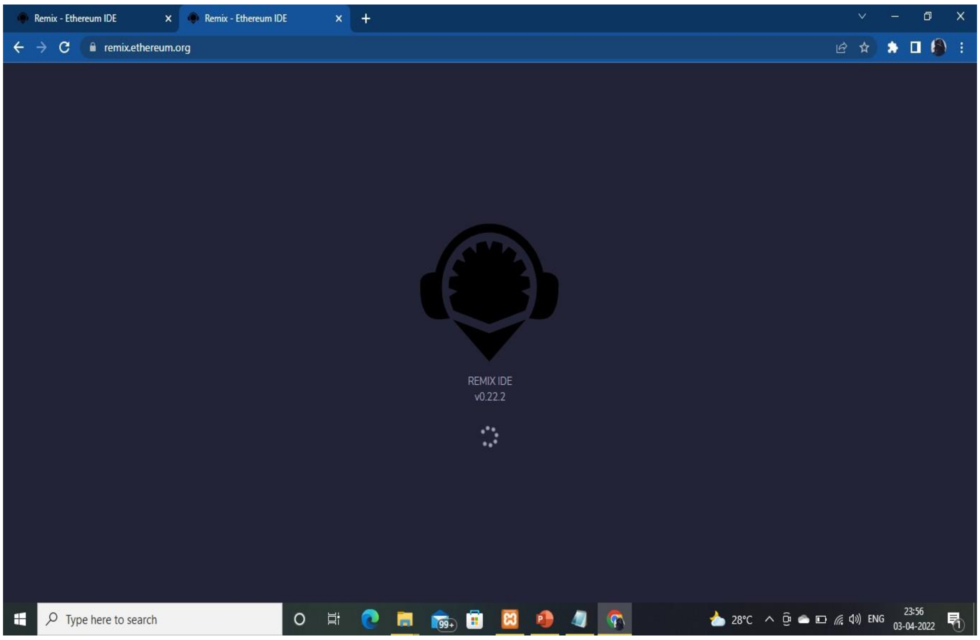
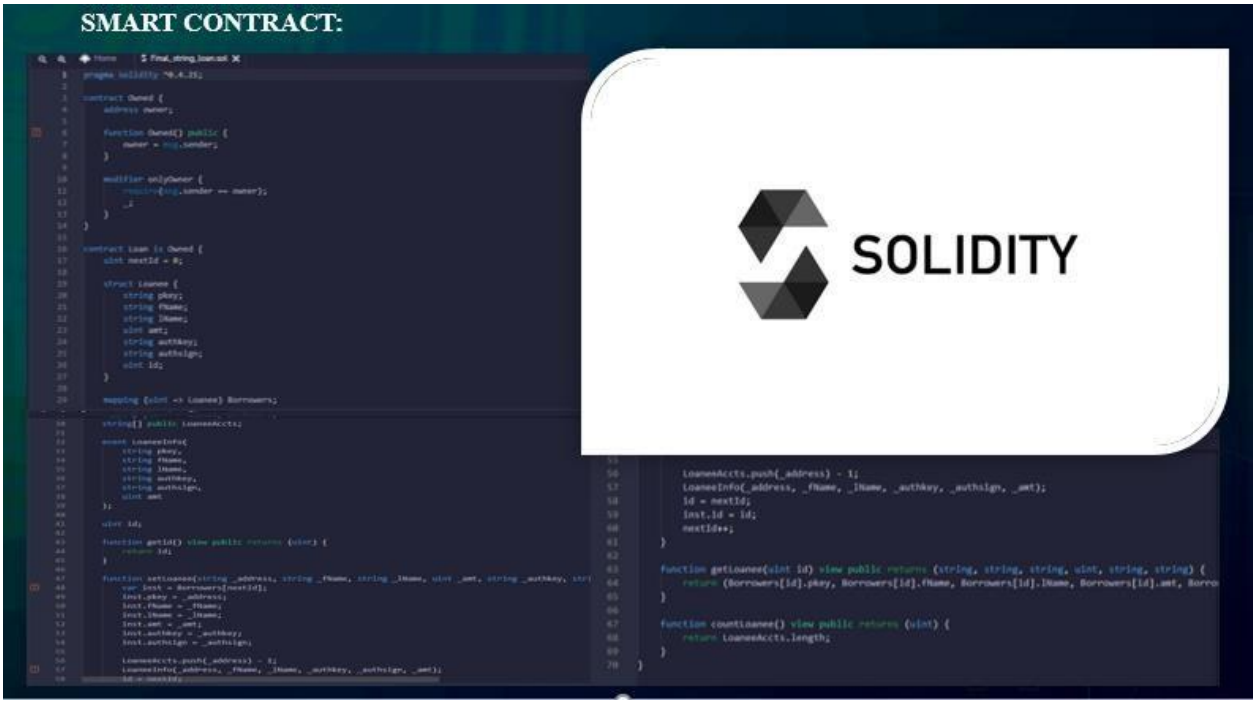


Figure 4.44 The remix ide appears create a smart contract to compile and deploy

4.6.6 PHASE 6: Create a Smart contract using solidity

- Solidity is an object-oriented, high-level programming language used to create smart contracts that automate transactions on the blockchain..
- Solidity is a curly-bracket language designed to target the Ethereum Virtual Machine (EVM). It is influenced by C++, Python and Javascript.
- A smart Contract for sanctioning Loan is created


SMART CONTRACT:

```
pragma solidity ^0.4.25;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Loan is Owned {
    uint nextId = 0;

    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
    );

    uint id;

    function getId() view public returns (uint) {
        return id;
    }

    function setLoanee(string _address, string _fName, string _lName, uint _amt, string _authkey, stri
        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;
```

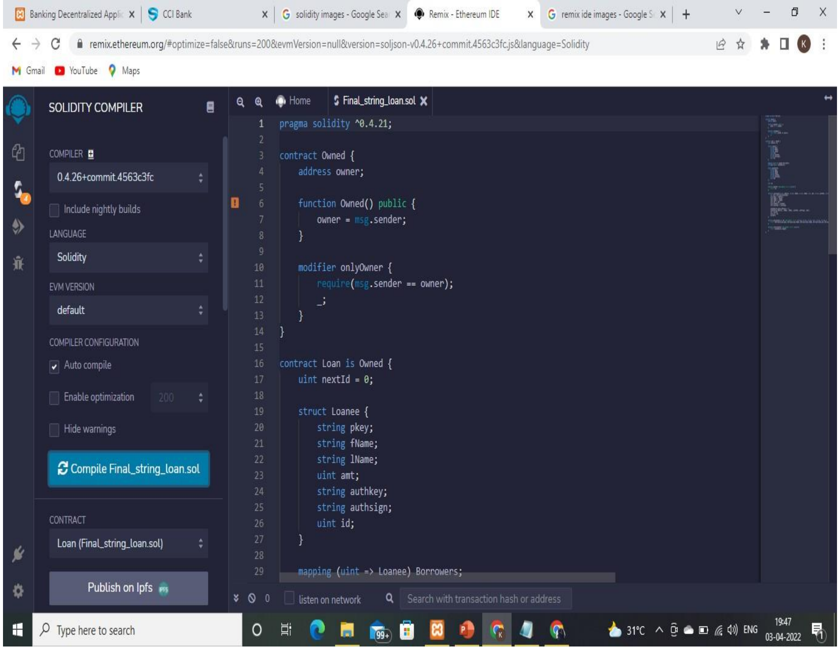
```
        LoaneeAccts.push(_address) - 1;
        LoaneeInfo(_address, _fName, _lName, _authkey, _authsign, _amt);
        id = nextId;
        inst.id = id;
        nextId++;
    }

    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, Borro
    }

    function countLoanee() view public returns (uint) {
        return LoaneeAccts.length;
    }
}
```

Figure 4.45 Smart Contract using solidity code

4.6.7 PHASE 7: TESTING SOLIDITY SMART CONTRACT IN REMIX IDE

- Remix, more commonly known as Remix IDE, is an open-source Ethereum IDE can use to write, compile and debug Solidity code As such, Remix can be a hugely important tool in Web3 and dapps development.
- The Smart Contract is Compiled and Deployed for Sanctioning a loan using web3 provider, injected Web3
- Both Injected Web3 and Web3 Provider require the use of an external tool.An external tool for Injected provider is Metamask. Some external tools used with Web3provider are a Truffle Ganache-CLI, Hardhat node, or an Ethereum node itself.

WEB3 PROVIDER



COMPILER: 0.4.26+commit.4563c3fc

□ Include nightly builds

LANGUAGE: Solidity

EVM VERSION: default

COMPILER CONFIGURATION

☑ Auto compile

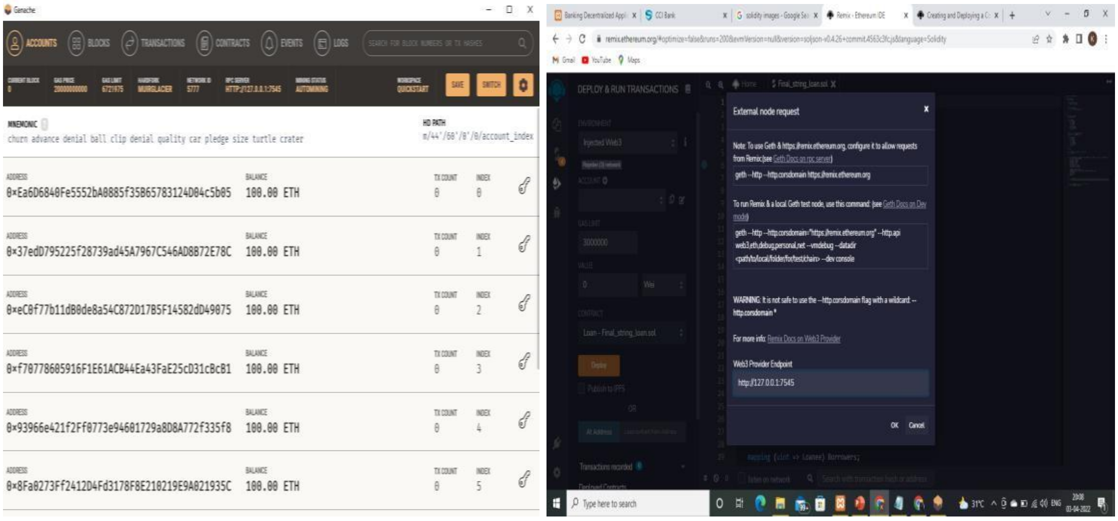
□ Enable optimization 200

□ Hide warnings

CONTRACT: Loan (Final_string_loan.sol)

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Loan is Owned {
    uint nextId = 0;

    struct Loanee {
        string pkey;
        string fName;
        string lName;
        uint amt;
        string authkey;
        string authsign;
        uint id;
    }

    mapping (uint => Loanee) Borrowers;
```

Figure 4.46 Compile the smart contract



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
0	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:7545	AUTOMINING	QUICKSTART

MNEMONIC: churn advance denial ball clip denial quality car pledge size turtle crater

HD PATH: m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×Ea6D6840Fe5552bA0885f35B65783124D04c5b05	100.00 ETH	0	0
0×37edD795225f28739ad45A7967C546AD8B72E78C	100.00 ETH	0	1
0×eC0f77b11dB0de8a54C872D17B5F14582dD49075	100.00 ETH	0	2
0×f70778605916F1E61ACB44Ea43FaE25cD31cBcB1	100.00 ETH	0	3
0×93966e421f2Ff0773e94601729a8D8A772f335f8	100.00 ETH	0	4
0×8Fa0273Ff2412D4Fd3178F8E210219E9A821935C	100.00 ETH	0	5

External node request

Note: To use Geth & https://remix.ethereum.org, configure it to allow requests from Remix:(see Geth Docs on rpc server)

```
geth --http --http.corsdomain https://remix.ethereum.org
```

To run Remix & a local Geth test node, use this command: (see Geth Docs on Dev mode)

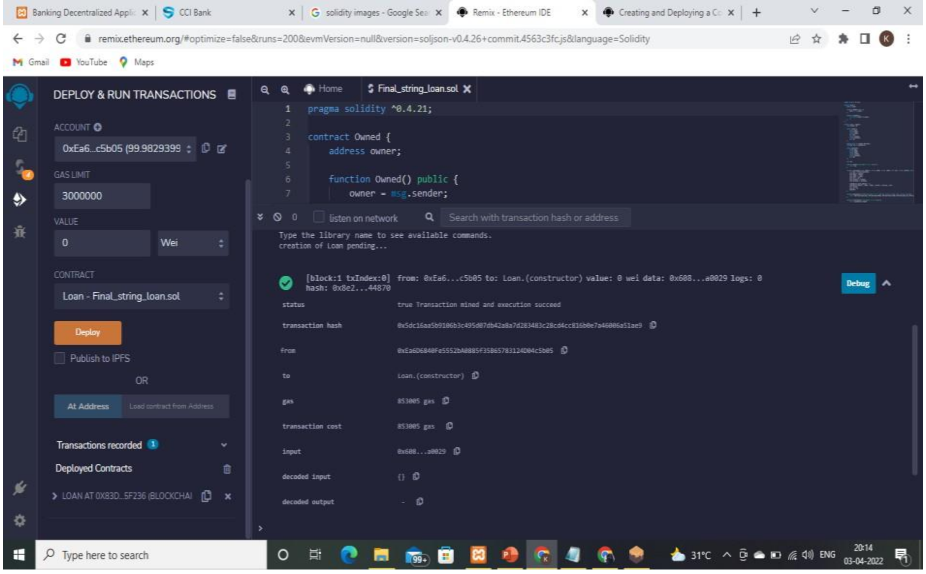
```
geth --http --http.corsdomain="https://remix.ethereum.org" --http.api web3,eth,debug,personal,net --vmdebug --datadir <path/to/local/folder/for/test/chain> --dev console
```

WARNING: It is not safe to use the --http.corsdomain flag with a wildcard: --http.corsdomain *

For more info: Remix Docs on Web3 Provider

Web3 Provider Endpoint: http://127.0.0.1:7545

Figure 4.47 Linking the url of ganache with web3 provider


DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 1
Deployed Contracts
LOAN AT 0X83D...5F236 (BLOCKCHAI
Home
Final_string_loan.sol
listen on network
Search with transaction hash or address
Debug

ACCOUNT: 0xEa6...c5b05 (99.9829399

GAS LIMIT: 3000000

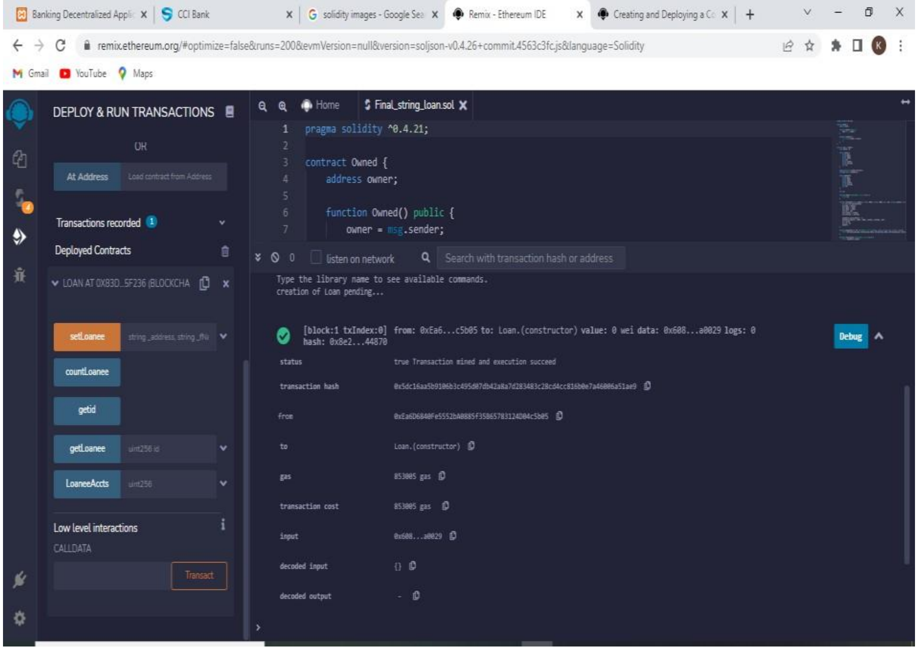
VALUE: 0 Wei

CONTRACT: Loan - Final_string_loan.sol

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
Type the library name to see available commands.
creation of Loan pending...

[block:1 txIndex:0] from: 0xEa6...c5b05 to: Loan.(constructor) value: 0 wei data: 0x608...a0029 logs: 0
hash: 0x8e2...44870
status               true Transaction mined and execution succeed
transaction hash     0x5dc16aa5b9106b3c495d07db42a8a7d283483c28cd4cc816b0e7a46006a51ae9
from                 0xEa6D6840Fe5552bA0885F35B65783124D04c5b05
to                   Loan.(constructor)
gas                  853005 gas
transaction cost     853005 gas
input                0x608...a0029
decoded input        {}
decoded output        -
```

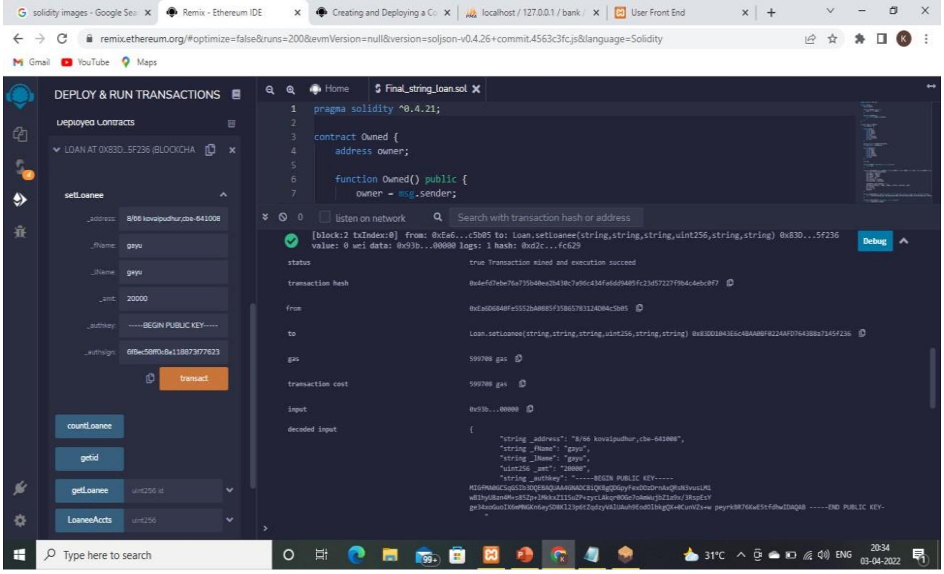
Figure 4.48 Deployed contract successfully



```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
Type the library name to see available commands.
creation of Loan pending...

[block:1 txIndex:0] from: 0xEa6...c5b05 to: Loan.(constructor) value: 0 wei data: 0x608...a0029 logs: 0
hash: 0x8e2...44870
status               true Transaction mined and execution succeed
transaction hash     0x5dc16aa5b9106b3c495d07db42a8a7d283483c28cd4cc816b0e7a46006a51ae9
from                 0xEa6D6840Fe5552bA0885F35B65783124D04c5b05
to                   Loan.(constructor)
gas                  853005 gas
transaction cost     853005 gas
input                0x608...a0029
decoded input        {}
decoded output        -
```

Figure 4.49 Giving loan details


solidity images - Google Sea
Remix - Ethereum IDE
Creating and Deploying a Co
localhost / 127.0.0.1 / bank /
User Front End
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity
Gmail
YouTube
Maps
DEPLOY & RUN TRANSACTIONS
Deployed Contracts
LOAN AT 0X83D...5F236 (BLOCKCHA
setLoanee
transact
countLoanee
getid
getLoanee
LoaneeAccts
Home
Final_string_loan.sol
listen on network
Search with transaction hash or address
Debug
Type here to search
31°C
ENG
20:34
03-04-2022

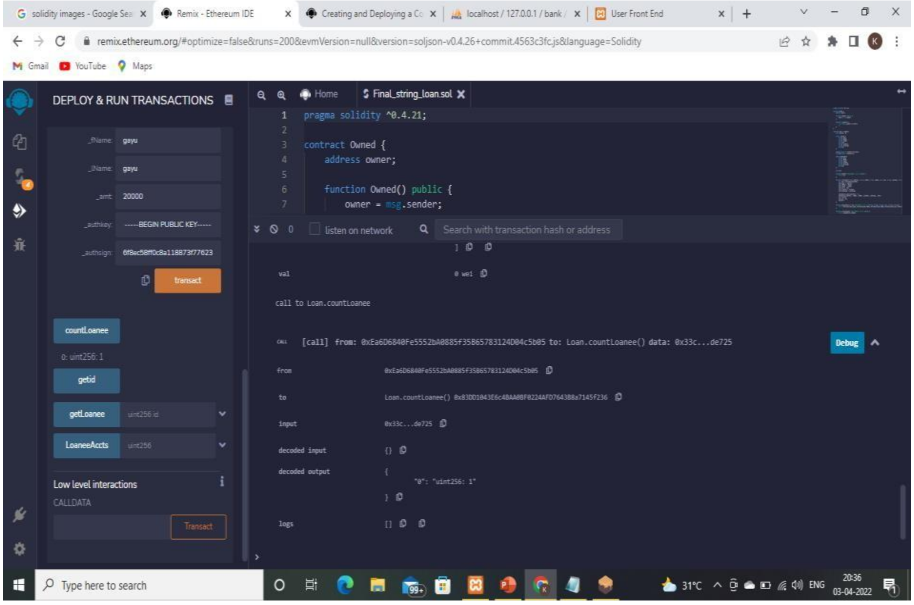
_address: 8/66 kovaipudhur,cbe-641008
_fName: gayu
_lName: gayu
_amt: 20000
_authkey: -----BEGIN PUBLIC KEY-----
_authsign: 6f8ec58ff0c8a118873f77623

getLoanee: uint256 id
LoaneeAccts: uint256

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
[block:2 txIndex:0] from: 0xEa6...c5b05 to: Loan.setLoanee(string,string,string,uint256,string,string) 0x83D...5f236
value: 0 wei data: 0x93b...00000 logs: 1 hash: 0xd2c...fc629
status              true Transaction mined and execution succeed
transaction hash    0x4efd7ebe76a735b40ea2b430c7a96c434fa6dd9405fc23d57227f9b4c4ebc0f7
from                0xEa6D6840Fe5552bA0885f35B65783124D04c5b05
to                  Loan.setLoanee(string,string,string,uint256,string,string) 0x83DD1043E6c4BAA08F0224AFD764388a7145f236
gas                 599708 gas
transaction cost    599708 gas
input               0x93b...00000
decoded input       {
                        "string _address": "8/66 kovaipudhur,cbe-641008",
                        "string _fName": "gayu",
                        "string _lName": "gayu",
                        "uint256 _amt": "20000",
                        "string _authkey": "-----BEGIN PUBLIC KEY-----
                    MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDrnAxQRsN3vusLMi
                    wB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycLAkqr0OGe7oAmWujbZ1a9x/3RspEsY
                    ge34xoGuoIX6mMNGKn6aySD8Kl23p6tZqdzyVAlUAuh9EodOlbkgQX+0CunVZs+w peyrkBR76KwE5tfdhwIDAQAB -----END PUBLIC KEY-
```

Figure 4.50 Set loan transaction


solidity images - Google Sea
Remix - Ethereum IDE
Creating and Deploying a Co
localhost / 127.0.0.1 / bank /
User Front End
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity
Gmail
YouTube
Maps
DEPLOY & RUN TRANSACTIONS
transact
countLoanee
getid
getLoanee
LoaneeAccts
Low level interactions
CALLDATA
Transact
Home
Final_string_loan.sol
listen on network
Search with transaction hash or address
Debug
Type here to search
31°C
ENG
20:36
03-04-2022

_fName: gayu
_lName: gayu
_amt: 20000
_authkey: -----BEGIN PUBLIC KEY-----
_authsign: 6f8ec58ff0c8a118873f77623

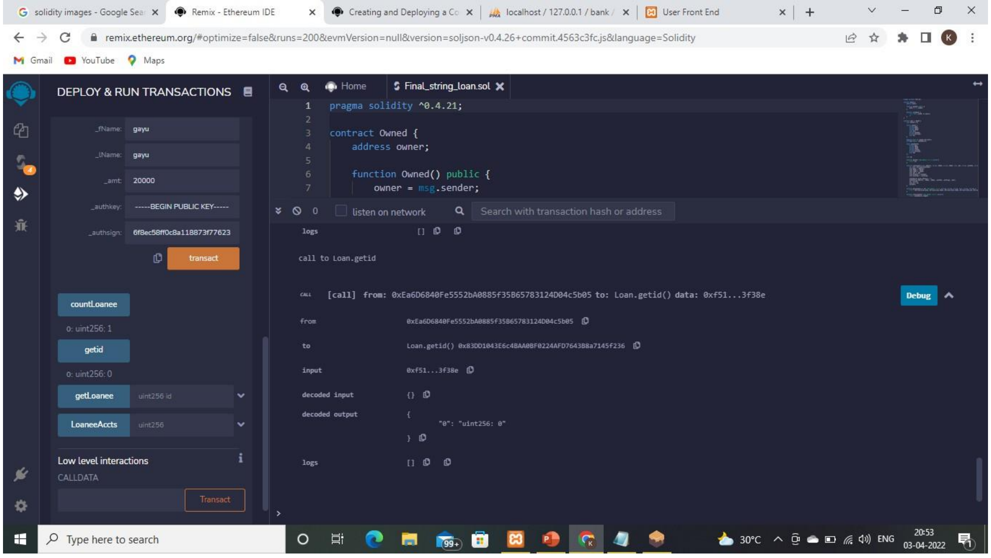
countLoanee
0: uint256: 1

getLoanee: uint256 id
LoaneeAccts: uint256

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
                    ]
val                 0 wei

call to Loan.countLoanee

CALL  [call] from: 0xEa6D6840Fe5552bA0885f35B65783124D04c5b05 to: Loan.countLoanee() data: 0x33c...de725
from                0xEa6D6840Fe5552bA0885f35B65783124D04c5b05
to                  Loan.countLoanee() 0x83DD1043E6c4BAA08F0224AFD764388a7145f236
input               0x33c...de725
decoded input       {}
decoded output      {
                        "0": "uint256: 1"
                    }
logs                []
```

Figure 4.51 Count loan



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity

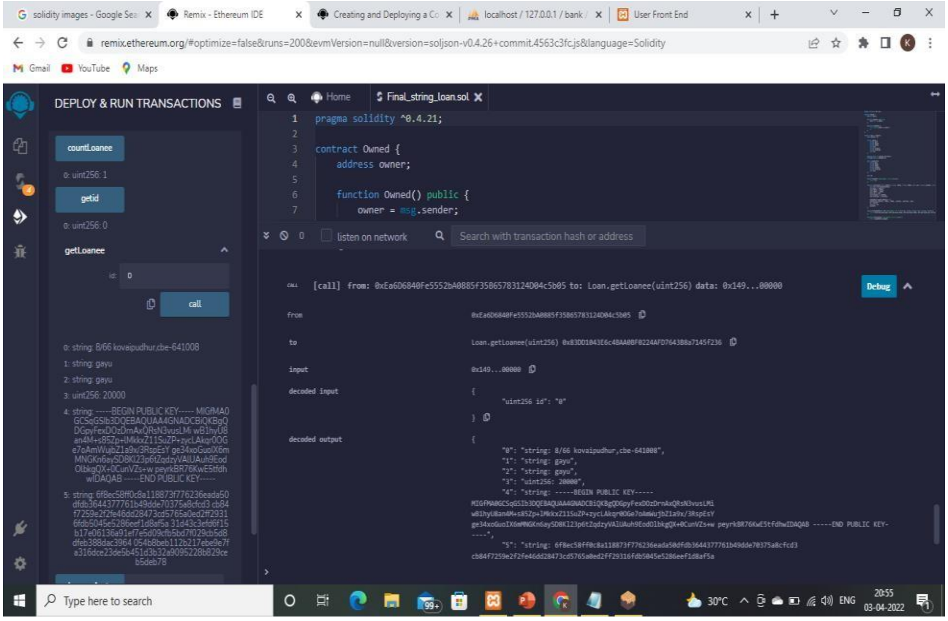
_fName: gayu
_lName: gayu
_amt: 20000
_authkey: -----BEGIN PUBLIC KEY-----
_authsign: 6f8ec58ff0c8a118873f77623

countLoanee
0: uint256: 1
getid
0: uint256: 0
getLoanee uint256 id
LoaneeAccts uint256

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
logs                 []

call to Loan.getid

[call] from: 0xEa6D6840Fe5552bA0885f35B65783124D04c5b05 to: Loan.getid() data: 0xf51...3f38e

from             0xEa6D6840Fe5552bA0885f35B65783124D04c5b05
to               Loan.getid() 0x83DD1043E6c4BAA0BF0224AFD7643B8a7145f236
input            0xf51...3f38e
decoded input    {}
decoded output   {
                        "0": "uint256: 0"
                 }
logs             []
```

Figure 4.52 Get id



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity

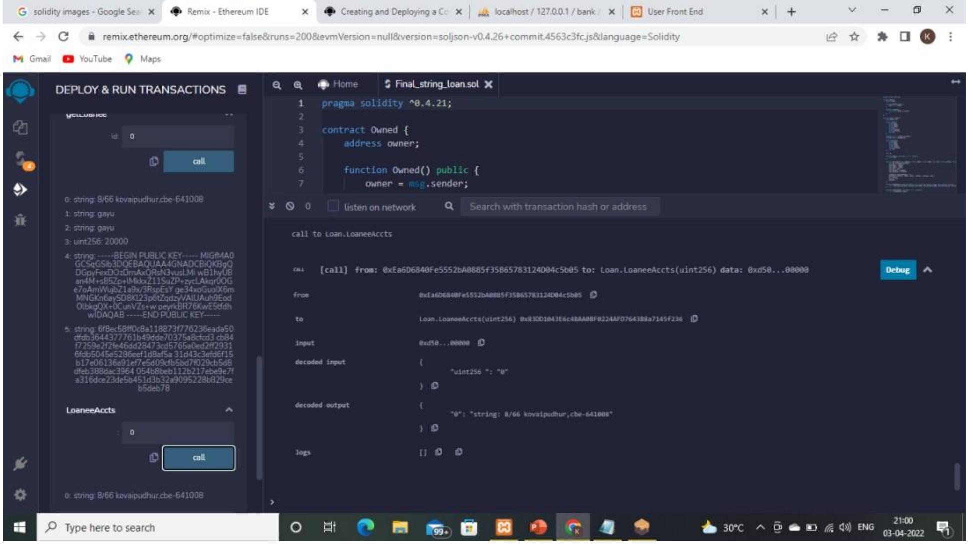
countLoanee
0: uint256: 1
getid
0: uint256: 0
getLoanee
id: 0

0: string: 8/66 kovaipudhur,cbe-641008
1: string: gayu
2: string: gayu
3: uint256: 20000
4: string: -----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDmAxQRsN3vusLMi wB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycLAkqr0OGe7oAmWujbZ1a9x/3RspEsY ge34xoGuoIX6mMNGKn6aySD8Kl23p6tZqdzyVAlUAuh9EodOlbkgQX+0CunVZs+w peyrkBR76KwE5tfdhwIDAQAB -----END PUBLIC KEY-----
5: string: 6f8ec58ff0c8a118873f776236eada50dfdb3644377761b49dde70375a8cfcd3 cb84f7259e2f2fe46dd28473cd5765a0ed2ff29316fdb5045e5286eef1d8af5a 31d43c3efd6f15b17e06136a91ef7e5d09cfb5bd7f029cb5d8dfeb388dac3964 054b8beb112b217ebe9e7fa316dce23de5b451d3b32a9095228b829ce b5deb78

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
[call] from: 0xEa6D6840Fe5552bA0885f35B65783124D04c5b05 to: Loan.getLoanee(uint256) data: 0x149...00000

from             0xEa6D6840Fe5552bA0885f35B65783124D04c5b05
to               Loan.getLoanee(uint256) 0x83DD1043E6c4BAA0BF0224AFD7643B8a7145f236
input            0x149...00000
decoded input    {
                        "uint256 id": "0"
                 }
decoded output   {
                        "0": "string: 8/66 kovaipudhur,cbe-641008",
                        "1": "string: gayu",
                        "2": "string: gayu",
                        "3": "uint256: 20000",
                        "4": "string: -----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDrnAxQRsN3vusLMi
wB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycLAkqr0OGe7oAmWujbZ1a9x/3RspEsY
ge34xoGuoIX6mMNGKn6aySD8Kl23p6tZqdzyVAlUAuh9EodOlbkgQX+0CunVZs+w peyrkBR76KwE5tfdhwIDAQAB -----END PUBLIC KEY-
----",
                        "5": "string: 6f8ec58ff0c8a118873f776236eada50dfdb3644377761b49dde70375a8cfcd3
cb84f7259e2f2fe46dd28473cd5765a0ed2ff29316fdb5045e5286eef1d8af5a
```

Figure 4.53 Call getloan



solidity images - Google Sea
Remix - Ethereum IDE
Creating and Deploying a Co
localhost / 127.0.0.1 / bank /
User Front End
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity
Gmail
YouTube
Maps
DEPLOY & RUN TRANSACTIONS
Home
Final_string_loan.sol
listen on network
Search with transaction hash or address
Debug
Type here to search
30°C
ENG
21:00
03-04-2022

getLoanee

id: 0

call

0: string: 8/66 kovaipudhur,cbe-641008

1: string: gayu

2: string: gayu

3: uint256: 20000

4: string: -----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDmAxQRsN3vusLMi wB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycLAkqr0OGe7oAmWujbZ1a9x/3RspEsY ge34xoGuolX6mMNGKn6aySD8Kl23p6tZqdzyVAlUAuh9EodOlbkgQX+0CunVZs+w peyrkBR76KwE5tfdhwIDAQAB -----END PUBLIC KEY-----

5: string: 6f8ec58ff0c8a118873f776236eada50dfdb3644377761b49dde70375a8cfcd3 cb84f7259e2f2fe46dd28473cd5765a0ed2ff29316fdb5045e5286eef1d8af5a 31d43c3efd6f15b17e06136a91ef7e5d09cfb5bd7f029cb5d8dfeb388dac3964 054b8beb112b217ebe9e7fa316dce23de5b451d3b32a9095228b829ceb5deb78

LoaneeAccts

: 0

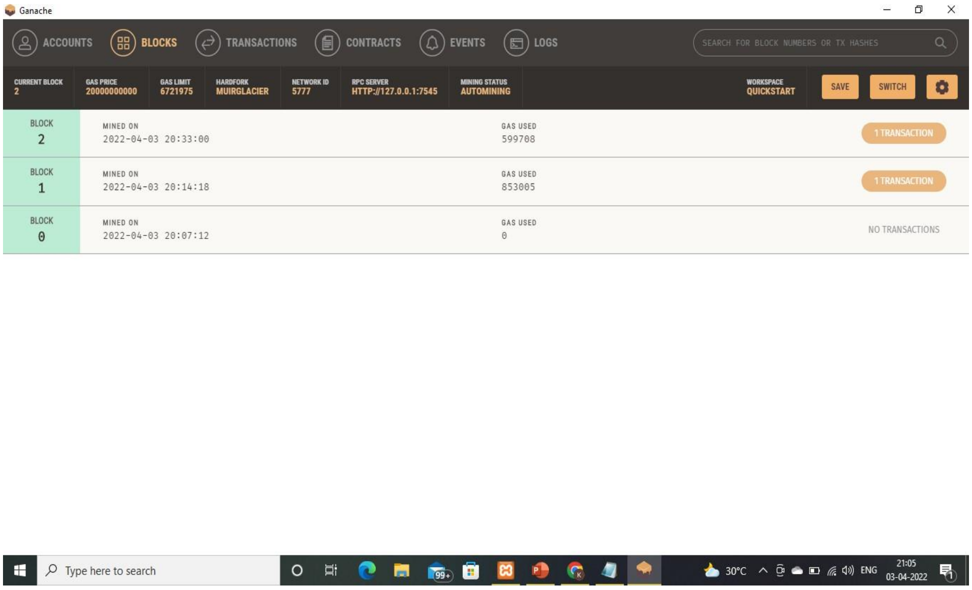
call

0: string: 8/66 kovaipudhur,cbe-641008

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
```

```
call to Loan.LoaneeAccts

CALL  [call] from: 0xEa6D6840Fe5552bA0885f35B65783124D04c5b05 to: Loan.LoaneeAccts(uint256) data: 0xd50...00000

from             0xEa6D6840Fe5552bA0885f35B65783124D04c5b05
to               Loan.LoaneeAccts(uint256) 0x83DD1043E6c48AA0BF0224AFD764388a7145f236
input            0xd50...00000
decoded input    {
                        "uint256 ": "0"
                 }
decoded output   {
                        "0": "string: 8/66 kovaipudhur,cbe-641008"
                 }
logs             []
```

Figure 4.54 Loan accounts



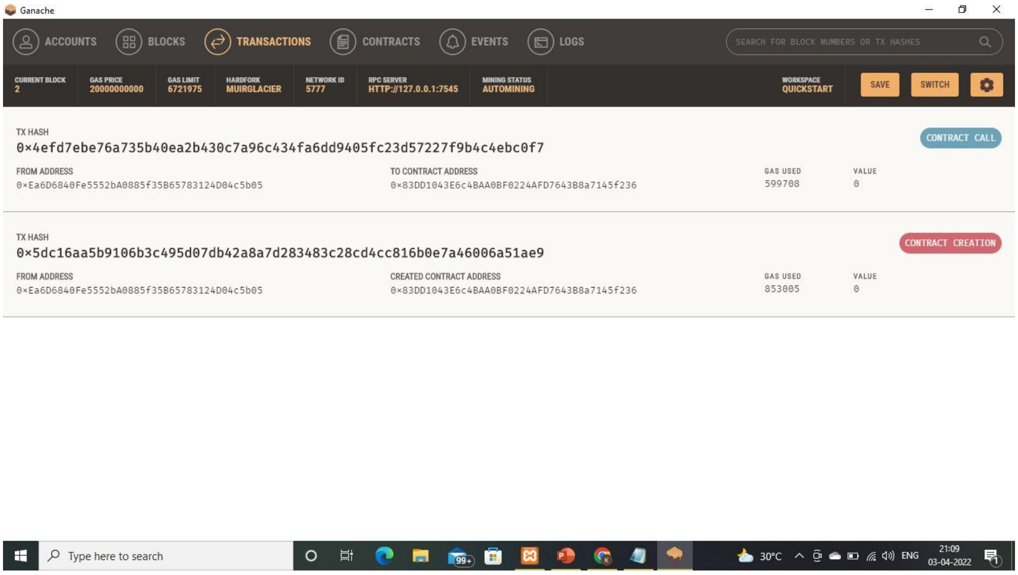
CURRENT BLOCK 2 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MUIRGLACIER | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE QUICKSTART

BLOCK 2 — MINED ON 2022-04-03 20:33:00 — GAS USED 599708 — 1 TRANSACTION

BLOCK 1 — MINED ON 2022-04-03 20:14:18 — GAS USED 853005 — 1 TRANSACTION

BLOCK 0 — MINED ON 2022-04-03 20:07:12 — GAS USED 0 — NO TRANSACTIONS

Figure 4.55 Blocks are displayed in the ganache



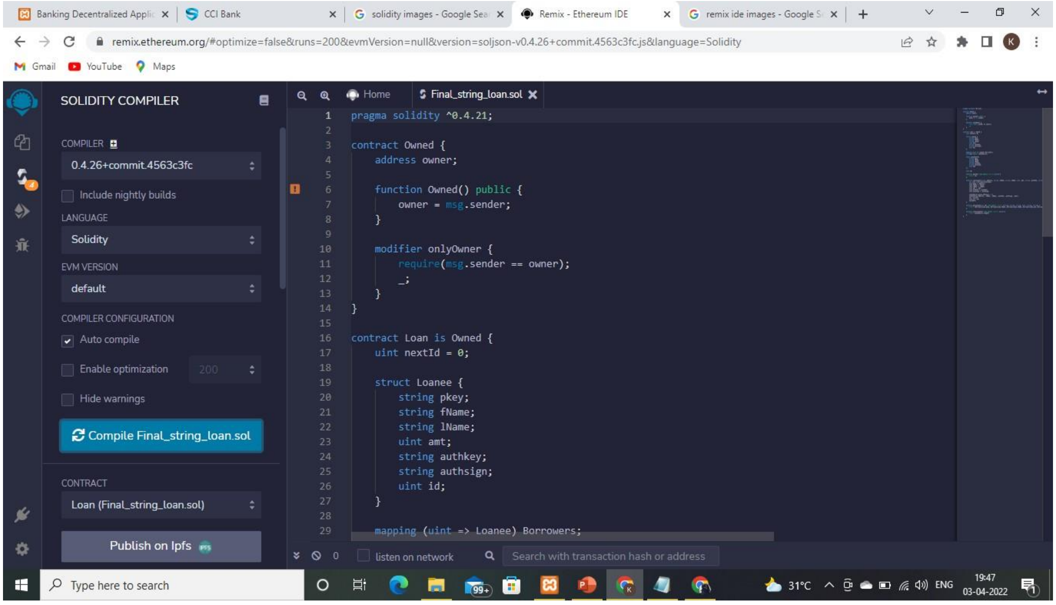
CURRENT BLOCK 2
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MUIRGLACIER
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE QUICKSTART

TX HASH
0×4efd7ebe76a735b40ea2b430c7a96c434fa6dd9405fc23d57227f9b4c4ebc0f7
CONTRACT CALL
FROM ADDRESS 0×Ea6D6840Fe5552bA0885f35B65783124D04c5b05
TO CONTRACT ADDRESS 0×83DD1043E6c4BAA0BF0224AFD7643B8a7145f236
GAS USED 599708
VALUE 0

TX HASH
0×5dc16aa5b9106b3c495d07db42a8a7d283483c28cd4cc816b0e7a46006a51ae9
CONTRACT CREATION
FROM ADDRESS 0×Ea6D6840Fe5552bA0885f35B65783124D04c5b05
CREATED CONTRACT ADDRESS 0×83DD1043E6c4BAA0BF0224AFD7643B8a7145f236
GAS USED 853005
VALUE 0

Figure 4.56 Transactions in ganache

INJECTED WEB3



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity

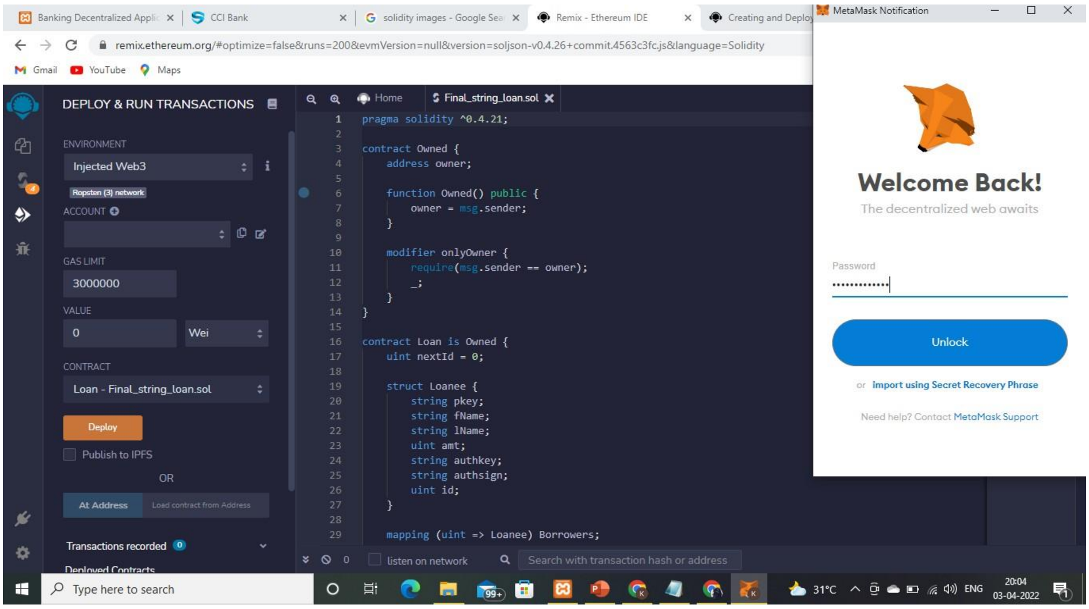
SOLIDITY COMPILER

COMPILER
0.4.26+commit.4563c3fc
□ Include nightly builds
LANGUAGE
Solidity
EVM VERSION
default
COMPILER CONFIGURATION
☑ Auto compile
□ Enable optimization 200
□ Hide warnings
Compile Final_string_loan.sol
CONTRACT
Loan (Final_string_loan.sol)
Publish on Ipfs

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Loan is Owned {
    uint nextId = 0;

    struct Loanee {
        string pkey;
        string fName;
        string lName;
        uint amt;
        string authkey;
        string authsign;
        uint id;
    }

    mapping (uint => Loanee) Borrowers;
```

0 □ listen on network Search with transaction hash or address

Figure 4.57 Compile the smart contract



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Injected Web3 (Ropsten (3) network)

ACCOUNT:

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: Loan - Final_string_loan.sol

Transactions recorded 0

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Loan is Owned {
    uint nextId = 0;

    struct Loanee {
        string pkey;
        string fName;
        string lName;
        uint amt;
        string authkey;
        string authsign;
        uint id;
    }

    mapping (uint => Loanee) Borrowers;
```

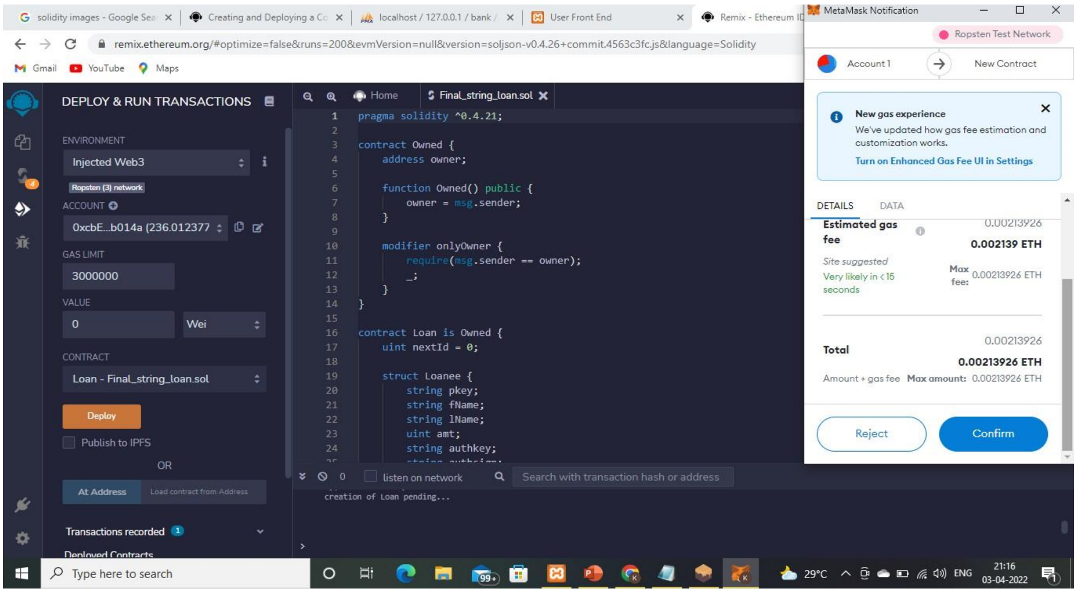
MetaMask Notification

Welcome Back!

The decentralized web awaits

Password: ••••••••••••••

Figure 4.58 Deploy a contract with inected web3 connecting metamask



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Injected Web3 (Ropsten (3) network)

ACCOUNT: 0xcbE...b014a (236.012377

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: Loan - Final_string_loan.sol

Transactions recorded 1

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Loan is Owned {
    uint nextId = 0;

    struct Loanee {
        string pkey;
        string fName;
        string lName;
        uint amt;
        string authkey;
```

```
creation of Loan pending...
```

MetaMask Notification

Ropsten Test Network

Account 1 → New Contract

New gas experience

We've updated how gas fee estimation and customization works.

Turn on Enhanced Gas Fee UI in Settings

DETAILS | DATA

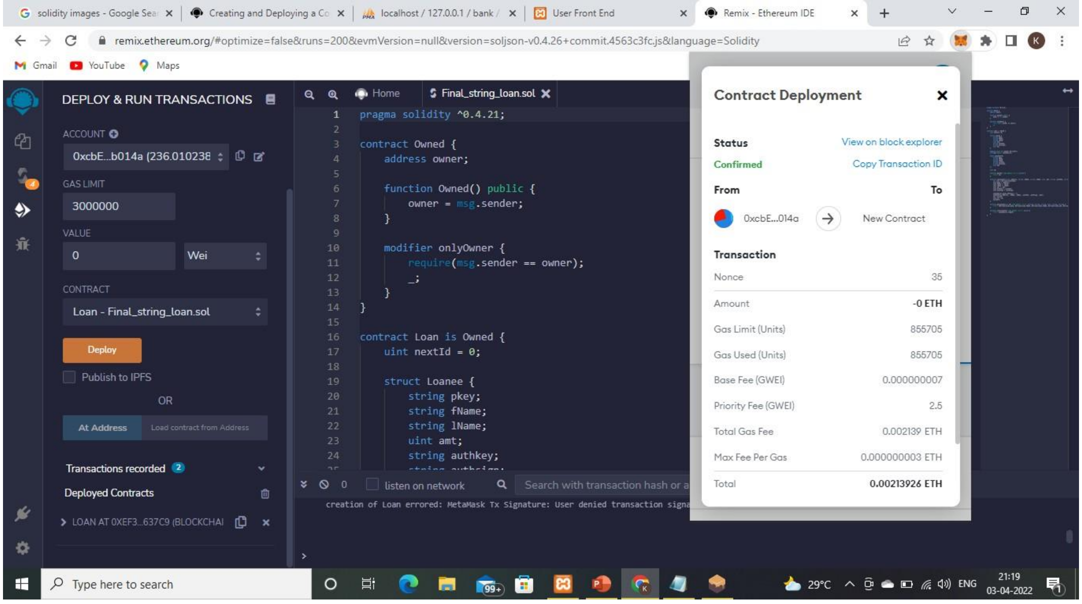
Estimated gas fee: 0.00213926 — 0.002139 ETH

Site suggested — Very likely in < 15 seconds — Max fee: 0.00213926 ETH

Total: 0.00213926 — 0.00213926 ETH

Amount + gas fee — Max amount: 0.00213926 ETH

Figure 4.59 Contract deployed with the gasprice



DEPLOY & RUN TRANSACTIONS

ACCOUNT: 0xcbE...b014a (236.010238

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: Loan - Final_string_loan.sol

Transactions recorded 2

Deployed Contracts

LOAN AT 0XEF3...637C9 (BLOCKCHAI

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Loan is Owned {
    uint nextId = 0;

    struct Loanee {
        string pkey;
        string fName;
        string lName;
        uint amt;
        string authkey;
```

```
creation of Loan errored: MetaMask Tx Signature: User denied transaction signa
```

Contract Deployment

Status: Confirmed

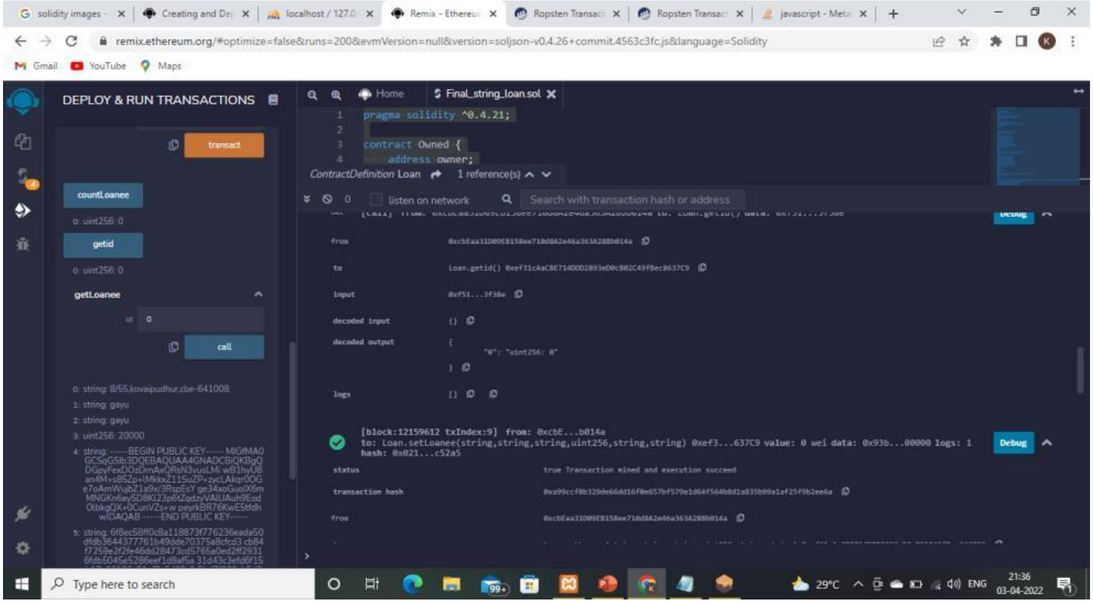
From: 0xcbE...014a → To: New Contract

Transaction

Nonce	35
Amount	-0 ETH
Gas Limit (Units)	855705
Gas Used (Units)	855705
Base Fee (GWEI)	0.000000007
Priority Fee (GWEI)	2.5
Total Gas Fee	0.002139 ETH
Max Fee Per Gas	0.000000003 ETH
Total	0.00213926 ETH

solidity images - Google Sea
Creating and Deploying a Co
localhost / 127.0.0.1 / bank /
User Front End
Remix - Ethereum IDE
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js&language=Solidity
Gmail
YouTube
Maps
Home
Final_string_loan.sol
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
listen on network
Search with transaction hash or a
View on block explorer
Copy Transaction ID
Type here to search
29°C
ENG
21:19
03-04-2022

Figure 4.60 Contract deployed successfully



countLoanee

0: uint256: 0

getid

0: uint256: 0

getLoanee

id: 0

0: string: 8/55,kovaipudhur,cbe-641008

1: string: gayu

2: string: gayu

3: uint256: 20000

4: string: -----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpyFexDOzDmAxQRsN3vusLMi wB1hyU8an4M+s85Zp+lMkkxZ11SuZP+zycLAkqr0OGe7oAmWujbZ1a9x/3RspEsY ge34xoGuolX6mMNGKn6eySD8Kl23p6tZqdzyVAlUAuh9EodOlbkgQX+0CunVZs+w peyrkBR76KwE5tfdhwIDAQAB -----END PUBLIC KEY-----

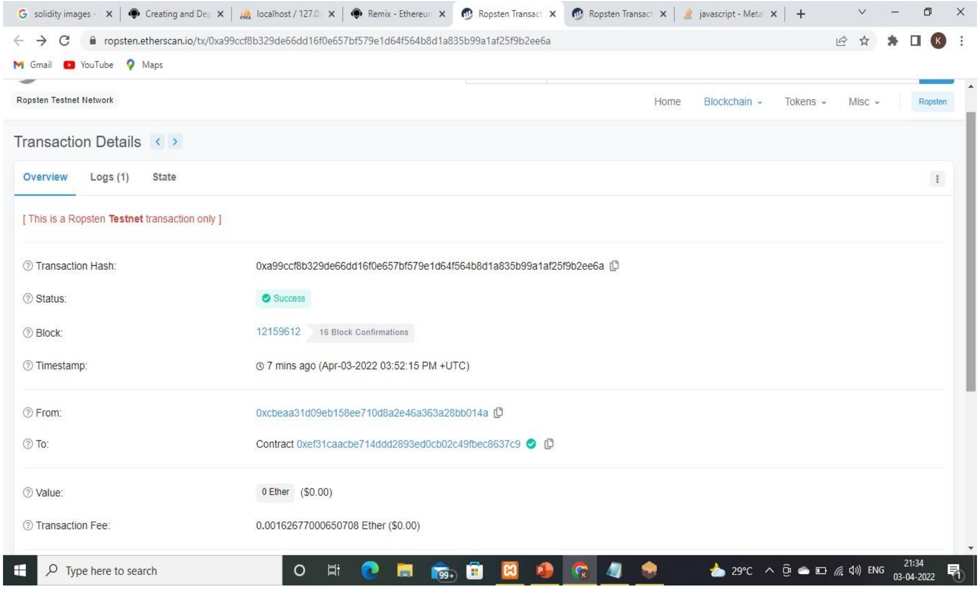
5: string: 6f8ec58ff0c8a118873f776236eada50dfdb3644377761b49dde70375a8cfcd3 cb84f7259e2f2fe46dd28473cd5765a0ed2ff2931 6fdb5045e5286eef1d8af5a 31d43c3efd6f15

```
pragma solidity ^0.4.21;

contract Owned {
    address owner;
```

ContractDefinition Loan 1 reference(s)

```
from            0xcbEaa31D09EB158ee710d8A2e46a363A288b014a
to              Loan.getid() 0xef31cAaCBE714000289 3eD0c802C49f8ec8637C9
input           0xf51...3f38e
decoded input   {}
decoded output  {
                    "0": "uint256: 0"
                }
logs            []

[block:12159612 txIndex:9]  from: 0xcbE...b014a
to: Loan.setLoanee(string,string,string,uint256,string,string) 0xef3...637C9 value: 0 wei data: 0x93b...00000 logs: 1
hash: 0x021...c52a5
status              true Transaction mined and execution succeed
transaction hash    0xa99ccf8b329de66dd16f0e657bf579e1d64f564b8d1a835b99a1af25f9b2ee6a
from                0xcbEaa31D09EB158ee710d8A2e46a363A288b014a
```

Figure 4.61 Set loan, count loan, get id, get loan, loan accounts



ropsten.etherscan.io/tx/0xa99ccf8b329de66dd16f0e657bf579e1d64f564b8d1a835b99a1af25f9b2ee6a

Ropsten Testnet Network

Transaction Details

[This is a Ropsten **Testnet** transaction only]

Transaction Hash: 0xa99ccf8b329de66dd16f0e657bf579e1d64f564b8d1a835b99a1af25f9b2ee6a

Status: Success

Block: 12159612 16 Block Confirmations

Timestamp: 7 mins ago (Apr-03-2022 03:52:15 PM +UTC)

From: 0xcbeaa31d09eb158ee710d8a2e46a363a28bb014a

To: Contract 0xef31caacbe714ddd2893ed0cb02c49fbec8637c9

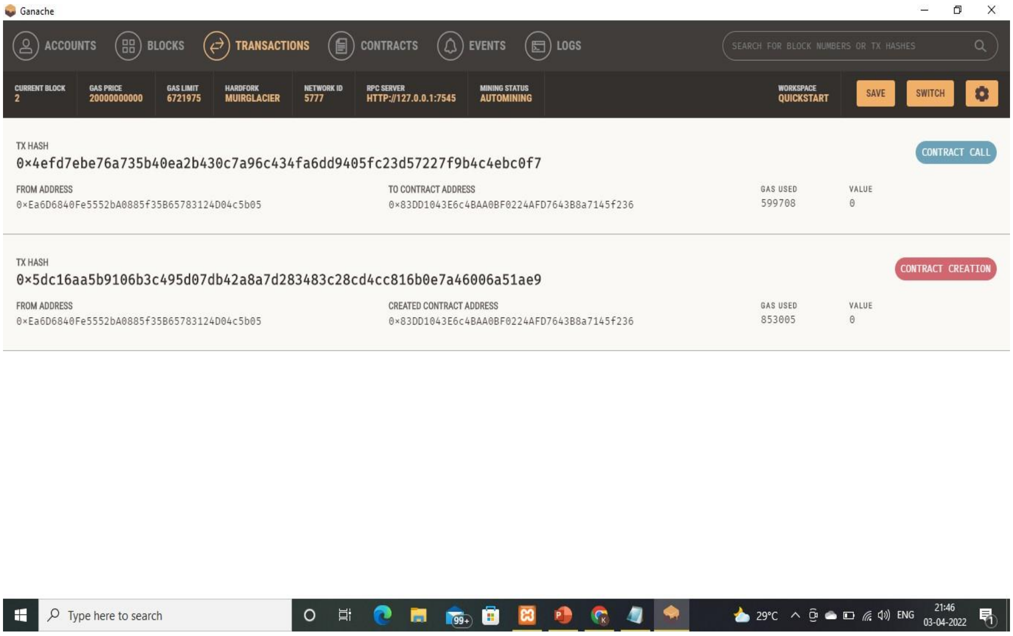
Value: 0 Ether ($0.00)

Transaction Fee: 0.00162677000650708 Ether ($0.00)

Figure 4.62 Viewetherscan outcome

4.6.8 Phase 8: Report of the transaction

This Report shows the transaction of WEB3 PROVIDER and INJECTED WEB3.



Ganache

CURRENT BLOCK 2 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MUIRGLACIER | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE QUICKSTART

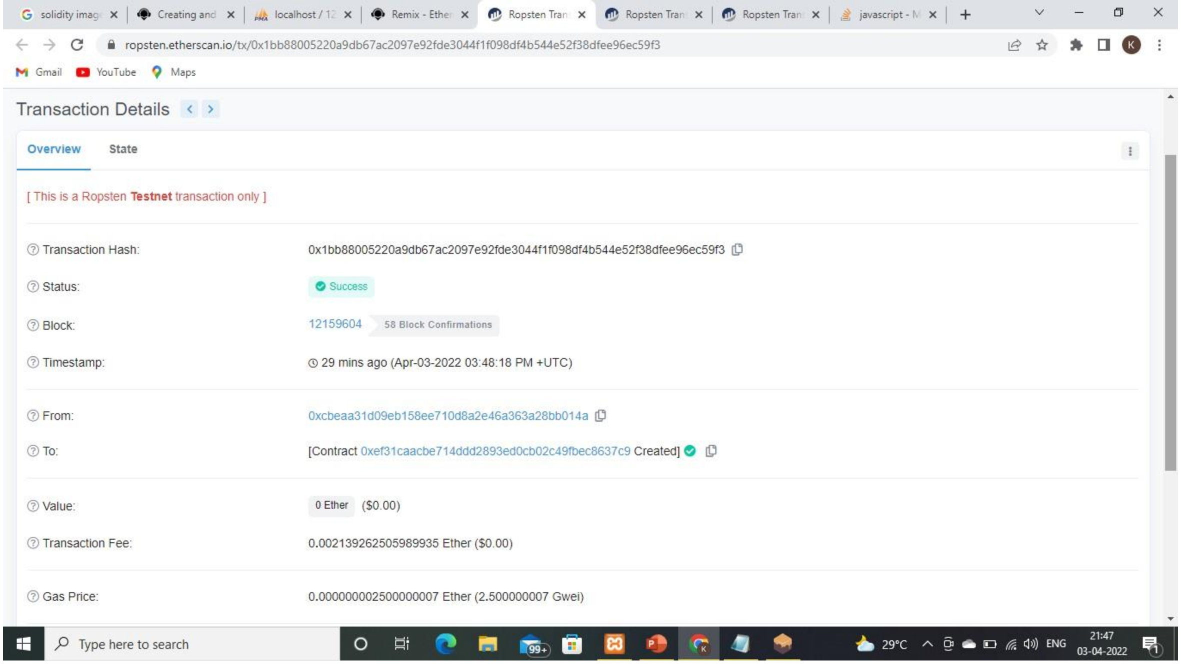
TX HASH
0×4efd7ebe76a735b40ea2b430c7a96c434fa6dd9405fc23d57227f9b4c4ebc0f7
CONTRACT CALL

FROM ADDRESS 0×Ea6D6840Fe5552bA0885f35B65783124D04c5b05
TO CONTRACT ADDRESS 0×83DD1043E6c4BAA0BF0224AFD7643B8a7145f236
GAS USED 599708
VALUE 0

TX HASH
0×5dc16aa5b9106b3c495d07db42a8a7d283483c28cd4cc816b0e7a46006a51ae9
CONTRACT CREATION

FROM ADDRESS 0×Ea6D6840Fe5552bA0885f35B65783124D04c5b05
CREATED CONTRACT ADDRESS 0×83DD1043E6c4BAA0BF0224AFD7643B8a7145f236
GAS USED 853005
VALUE 0

Figure 4.63 Web3 provider outcome



ropsten.etherscan.io/tx/0x1bb88005220a9db67ac2097e92fde3044f1f098df4b544e52f38dfee96ec59f3

Transaction Details

[This is a Ropsten **Testnet** transaction only]

Transaction Hash:	0x1bb88005220a9db67ac2097e92fde3044f1f098df4b544e52f38dfee96ec59f3
Status:	Success
Block:	12159604 58 Block Confirmations
Timestamp:	29 mins ago (Apr-03-2022 03:48:18 PM +UTC)
From:	0xcbeaa31d09eb158ee710d8a2e46a363a28bb014a
To:	[Contract 0xef31caacbe714ddd2893ed0cb02c49fbec8637c9 Created]
Value:	0 Ether ($0.00)
Transaction Fee:	0.002139262505989935 Ether ($0.00)
Gas Price:	0.000000002500000007 Ether (2.500000007 Gwei)

Figure 4.64 Injected web3 outcome viewetherscan

CHAPTER 5
CONCLUSION

Chapter 5

Conclusion

The Ethereum protocol, as the upgraded cryptocurrency, focuses not only on transactions andmining, but it also implements a nearly Turing-complete language on its blockchain, as well asa notable smart contract architecture. By merging the concept of decentralisation into web applications, Ethereum represented the beginning of the evolution from Web 2.0 to Web 3.0, the third generation of Internet-based services that collectively compose what could be called the intelligent web.

The appropriate approach is not to alter the entire banking sector since blockchain is not a cure for all of the problems that the banking system is now facing. Havingsaid that, blockchain is a perfect solution for ensuring data integrity and reducing fraud events.Regulators, payment processors, and auditors have real-time access to transactions in a private permissioned blockchain, making it much easier to detect any attempted fraud or attack. A blockchain-based technology reduces fraud by creating a network and securely communicatingtransaction details between institutions in real-time The solution enables institutions to protectvital client data while automatically recognizing any elements of a scam, major or little.

CHAPTER 6

SCOPE FOR FUTURE ENHANCEMENT

CHAPTER 6

Scope For Future Enhancement

The project has covered the objectives set for this work. But there is a lot of scope for future enhancements to this work. Further requirements and improvements can be easily done. The work can be Extended to explore all possible classification techniques available in software. Furthermore, banks' capacity to technologically react to the myriad new digital initiatives beingimplemented is currently restricting consumer experience in India. Banks may implement newdigital solutions more easily, quickly, and cheaply by integrating a blockchain technology solution with existing legacy systems.

CHAPTER 7
REFERENCES

Chapter 7

References

Web References

- The crypto wallet for Defi, Web3 Dapps and NFTs | MetaMask. (n.d.). Metamask. Retrieved May 9, 2022, from https://metamask.io/
- Community, N. (n.d.). Ganache CLI - Nethereum Documentation. Ganache-Cli. Retrieved May 9, 2022, from https://docs.nethereum.com/en/latest/ethereum-and-clients/ganache-cli/
- Ganache - Truffle Suite. (n.d.). Truffle. Retrieved May 9, 2022, from https://trufflesuite.com/ganache/
- Ethereum. (n.d.). Home. Ethereum.Org. Retrieved May 9, 2022, from https://www.ethereum.org/en/
- X. (n.d.). *GitHub - Xel/Blockchain-stuff: Blockchain and Crytocurrency Resources*. GitHub. Retrieved May 9, 2022, from https://github.com/Xel/Blockchain-stuff
- *Maze Found | Read the Docs*. (n.d.). Web3j. Retrieved May 9, 2022, from https://web3js.readthedocs.io/en/1.0/getting-started
- javascript:openWebLink('https://www.google.com/search?q=how+does+smart+contract+work&rlz=1C1VDKB_enIN969IN969&source=lnms&tbm=isch&sa=X&ved=2ahUKEwj_vcmBsfP3AhUzSmwGHfuNDTMQ_AUoAXoECAEQAw&biw=1366&bih=625&dpr=1#imgrc=YCQg-WKocHhTDM')
- javascript:openWebLink('https://www.google.com/search?q=blockchain+technology+images&tbm=isch&ved=2ahUKEwjCo-6CsfP3AhXWgGMGHb37DdIQ2-cCegQIABAA&oq=blockchain+technology&gs_lcp=CgNpbWcQARgBMggIABCABBCxAzIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDoECAAQGDoLCAAQgAQQsQMQgwE6CAgAELEDEIMBOgQIABBDOgoIABCxAxCDARBDOgcIABCxAxBDUKOPGliQ3xtgv_IbaAFwAHgEgAH0A4gBjCSSAQowLjI2LjQtMS4xmAEAoAEBqgELZ3dzLXdpei1pbWewAQDAAQE&sclient=img&ei=YlGKYsLqN9aBjuMPvfe3kA0&bih=625&biw=1366&rlz=1C1VDKB_enIN969IN969#imgrc=EmRK90jidO5lnM')

CHAPTER 8
APPENDIX

Chapter 8

Appendix

A.1 Backend : Solidity Code

```
pragma solidity ^0.4.21;

contract Owned {

    address owner;

    function Owned() public {

        owner = msg.sender;

    }

    modifier onlyOwner {

require(msg.sender == owner);

        _;

    }

}

contract Loan is Owned {

uintnextId = 0;

    struct Loan {

        string pkey;

        string fName;

        string lName;

uint amt;

        string authkey;
        string authsign;
uint id;
    }
   mapping (uint => Loanee) Borrowers;
string[] public LoanAccts;
    event LoanInfo(
        string pkey,
        string fName,
        string lName,
        string authkey,
        string authsign,
uint amt
    );
uint id;
    function getid() view public returns (uint) {
        return id;
    }
    function setLoan(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;

LoanAccts.push(_address) - 1;

LoanInfo(_address, _fName, _lName, _authkey, _authsign, _amt);

        id = nextId;

        inst.id = id;

nextId++;

    }

    function getLoan(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 countLoan() view public returns (uint) {

        return LoanAccts.length;

    }

}
```

Node.Js Code

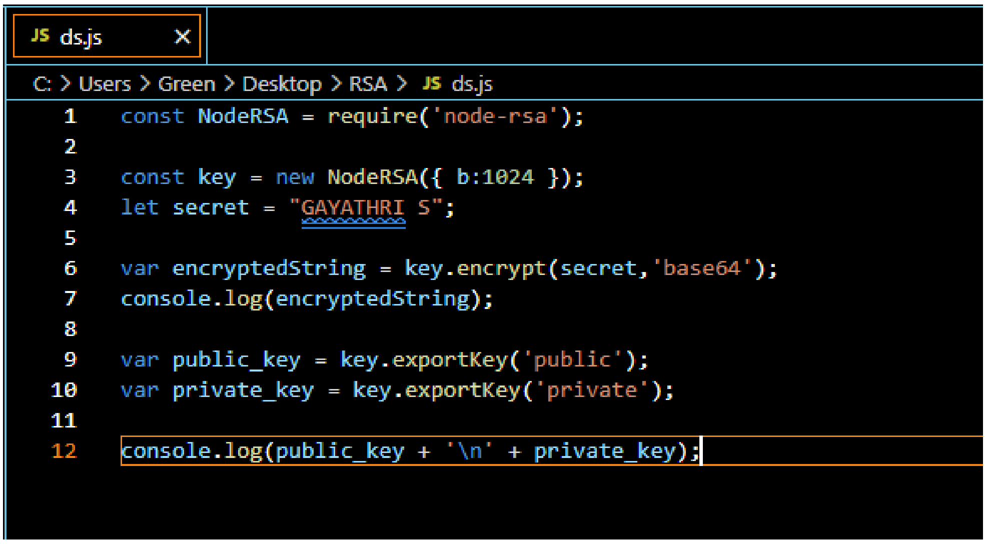
```
const NodeRSA = require('node-rsa');

const key = new NodeRSA({ b:1024 });

let secret = "GAYATHRI S";
```

```
var encryptedString = key.encrypt(secret,'base64');

console.log(encryptedString);

var public_key = key.exportKey('public');

var private_key = key.exportKey('private');

console.log(public_key + '\n' + private_key);
```

A.2 Frontend: Javascript Code

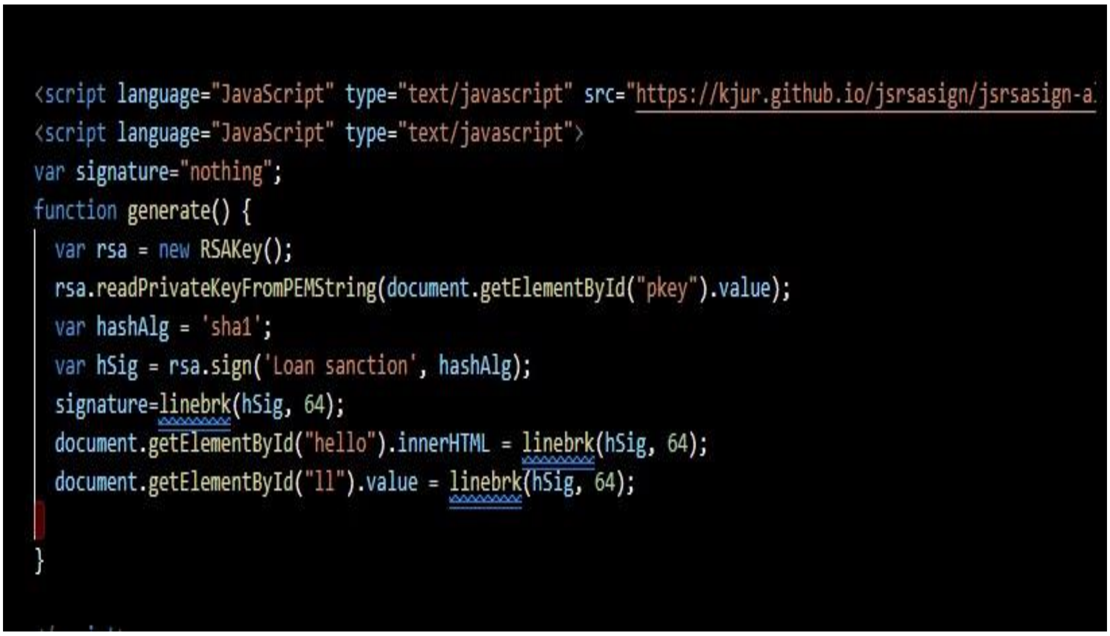
```
<scriptlanguage="JavaScript"type="text/javascript">

varsignature="nothing";

functiongenerate() {

  varrsa=newRSAKey();

  rsa.readPrivateKeyFromPEMString(document.getElementById("pkey").value);

  varhashAlg='sha1';

  varhSig=rsa.sign('Loan sanction', hashAlg);

  signature=linebrk(hSig, 64);

  document.getElementById("hello").innerHTML=linebrk(hSig, 64);

  document.getElementById("ll").value=linebrk(hSig, 64);

}
```

A.3 SCREENSHOTS



```javascript
const NodeRSA = require('node-rsa');

const key = new NodeRSA({ b:1024 });
let secret = "GAYATHRI S";

var encryptedString = key.encrypt(secret,'base64');
console.log(encryptedString);

var public_key = key.exportKey('public');
var private_key = key.exportKey('private');

console.log(public_key + '\n' + private_key);
```

Figure 8.1 Code for Public/Private key with RSA Algorithm



```javascript
<script language="JavaScript" type="text/javascript" src="https://kjur.github.io/jsrsasign/jsrsasign-a
<script language="JavaScript" type="text/javascript">
var signature="nothing";
function generate() {
  var rsa = new RSAKey();
  rsa.readPrivateKeyFromPEMString(document.getElementById("pkey").value);
  var hashAlg = 'sha1';
  var hSig = rsa.sign('Loan sanction', hashAlg);
  signature=linebrk(hSig, 64);
  document.getElementById("hello").innerHTML = linebrk(hSig, 64);
  document.getElementById("l1").value = linebrk(hSig, 64);

}
```

Figure 8.2 Generating Public/Private key in webpage by using R