DECENTRALIZED CHAT SYSTEM USING BLOCKCHAIN

Main Project Report

Submitted by

Ann Sarah Babu

Reg No : FIT20MCA-2029

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Computer Applications
Of
A P J Abdul Kalam Technological University

Focus on Excellence

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)®
ANGAMALY-683577, ERNAKULAM(DIST)
JULY 2022

DECLARATION

I, **Ann Sarah Babu** hereby declare that the report of this project work, submitted to the Department of Computer Applications, Federal Institute of Science and Technology (**FISAT**), Angamaly in partial fulfillment of the award of the degree of Master of Computer Applications is an authentic record of my original work.

The report has not been submitted for the award of any degree of this university or any other university.

Date : 12/07/2022 **Signature :** [illegible]

Place: Angamaly **Name :** ANN SARAH BABU

11TH July 2022

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr./Ms. ANN SARAH BABU (Reg. No. FIT20MCA-2029) has successfully completed his/her Main Project with the title "Decentralized Chat System Using Blockchain", in the Department of Computer Applications, FISAT, during the period from 30th March 2022 to 11th July 2022.

[illegible]

Dr DEEPA MARY MATHEWS

HEAD OF THE DEPARTMENT

FEDERAL INSTITUTE OF SCIENCE & TECHNOLOGY (FISAT) ANGAMALY ★ DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)®
ANGAMALY, ERNAKULAM-683577

DEPARTMENT OF COMPUTER APPLICATIONS

Focus on Excellence

CERTIFICATE

This is to certify that the project report titled **'Decentralized Chat System Using Blockchain'** submitted by **Ann Sarah Babu [Reg No: FIT20MCA-2029]** towards partial fulfillment of the requirements for the award of the degree of Master of Computer Applications is a record of bonafide work carried out by her during the year 2022.

[illegible]

Project Guide
Dr. Rakhi Venugopal

FEDERAL INSTITUTE OF SCIENCE & TECHNOLOGY (FISAT) ANGAMALY
DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

[illegible]

Head of the Department
Dr. Deepa Mary Mathews

Submitted for the viva-voice held on .... at ....

Examiner:

ACKNOWLEDGEMENT

I am deeply grateful to **Dr. Manoj George** , Principal, FISAT, Angamaly for providing me with adequate facilities, way and means by which I was able to complete my main project work and I express my sincere thanks to **Dr. C Sheela** ,Vice Principal FISAT, Angamaly.

My sincere thanks to **Dr. Deepa Mary Mathews**, HOD, Department of Computer Applications, FISAT, who had been a source of inspiration. I express heartiest thanks to **Dr. Rakhi Venugopal** my project guide for the encouragement and valuable suggestion . I express my heartiest gratitude to my scrum master **Dr. Shahana K U** and the faculty members in my department for their constant encouragement and never ending support throughout the project.I would also like to express my sincere gratitude to the lab faculty members for their guidance.

Finally I express my thanks to all my friends who gave me wealth of suggestions for successful completion of this project.

ABSTRACT

Blockchain technology has been seeing widespread interest as a means to ensure the integrity, confidentiality and availability of data in a trustless environment. They are designed to protect data from both internal and external cyberattacks by utilizing the aggregated power of the network to resist malicious efforts. This project aims to create decentralized messaging application utilizing the Ethereum protocol. The application will be able to send messages both securely and anonymously. Ethereum platform is used to deploy the blockchain network.

Decentralized application ensures that no network failure can occur due to central node failure. Blockchain will serve as an immutable ledger(any records that have the ability to remain unchanged) which allows messaging to take place in a decentralized manner. A decentralized application for communication and resource sharing is a need in today's world, where keeping data on a centralized server can be risky and costly experience. Together with Blockchain and Decentralization, a secure and reliable messaging application that overcomes the drawbacks of traditional messaging applications can be created.

Contents

1 INTRODUCTION **1**

2 PROOF OF CONCEPT **3**

2.1 Objectives 6

3 IMPLEMENTATION **7**

3.1 System Study 8

3.1.1 Existing System 8

3.1.2 Proposed System 9

3.2 Prerequisties 10

3.2.1 Ethereum 10

3.2.2 Truffle 11

3.2.3 Ganache 12

3.2.4 Smart Contracts 12

3.2.5 Metamask 14

3.2.6 Node js 14

3.2.7 React js 15

3.2.8 Web3 js 15

3.3 Working 16

3.4 Modules 17

3.4.1 Environment Setup 17

3.4.2 Deploy the first smart contract 17

3.4.3 Send Messages 20

4 RESULT ANALYSIS 24

5 CONCLUSION AND FUTURE SCOPE 27

5.1 Conclusion 27

5.2 Future Scope 29

6 APPENDIX 30

6.1 Source Code 30

6.1.1 Chat.js 30

6.1.2 ChatApp.sol 39

6.1.3 Migrations.sol 41

6.1.4 truffle-config.js 42

6.2 Screenshots 43

7 REFERENCES 47

Chapter 1

INTRODUCTION

A blockchain is a distributed database or ledger that is shared among the nodes of a computer network. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.Blockchain is a distributed network where all of the participants share the responsibility of running the network. Each network participant maintains a copy of the code and the data on the blockchain. All of this data is contained in bundles of records called ”blocks” which are ”chained together” to make up the blockchain. All of the nodes on the network ensure that this data is secure and unchangeable, unlike a centralized application where the code and data can be changed at any time. That’s what makes the blockchain so powerful!

One key difference between a typical database and a blockchain is how the data is structured. A blockchain collects information together in groups, known as blocks, that hold sets of information. Blocks have certain storage capacities and, when filled, are closed and linked to the previously filled block, forming a chain of data known as the blockchain. All new information that follows that freshly added block is compiled into a newly formed block that will then also be added to the chain once filled.

A database usually structures its data into tables, whereas a blockchain, as its name implies, structures its data into chunks (blocks) that are strung together. This data structure inherently makes an irreversible timeline of data when implemented in a decentralized nature. When a block is filled, it is set in stone and becomes a part of this timeline. Each block in the chain is given an exact timestamp when it is added to the chain.

The aim of the project is to develop a decentralized chat system using blockchain technology. If it is able to successfully implement a functioning Decentralized chat application the usage and deployment areas will be endless, it's something which solves and restores chat-privacy for people, and could also promise a solution for secure data transfer in Armed forces organizations.

The report is organized as follows: The next section has the Proof of Concept which is followed by the Implementation section. In the Implementation section, the workflow, prerequisites as well as the modules/sub-tasks are discussed. The Implementation section is followed by Result Analysis and then the Conclusion and References.

Chapter 2

PROOF OF CONCEPT

This chapter contains literature review related with different approaches that are currently used in blockchain technology and decentralization.In Blockchain technology overview[1]it is said that blockchains are tamper evident and tamper resistant digital ledgers implemented in a distributed fashion (i.e., without a central repository) and usually without a central authority (i.e., a bank, company, or government). At their basic level, they enable a community of users to record transactions in a shared ledger within that community, such that under normal operation of the blockchain network no transaction can be changed.

Block chain Guide[2],here it is discussed about what is a blockchain, how does it works, about its security.Blockchain in simple terms, a blockchain is a shared database or ledger. Pieces of data are stored in data structures known as blocks, and each node of the network has an exact replica of the entire database. Security is ensured since if somebody tries to edit or delete an entry in one copy of the ledger, the majority will not reflect this change and it will be rejected.One key difference between a typical database and a blockchain is how the data is structured.A database usually structures its data into tables, whereas a blockchain, as its name implies, structures its data into chunks (blocks) that are strung together.

In Ethereum Development[3] Ethereum is a decentralized blockchain platform that establishes a peer-to-peer network that securely executes and verifies application code, called smart contracts. Smart contracts allow participants to transact with each other without a trusted central authority.

In Blockchain Development with Ethereum and Solidity[4], Understand the basic philosophy behind the blockchain and distributed/decentralized applications. The basics of how a blockchain generally works, just enough to be able to use it as a development platform.

Decentralized Chat Application using Blockchain Technology[5],as we know traditional chat applications are centralized i.e., all the data is stored on a centralized server. Therefore major problem of this structure is, if the central server fails then whole network collapses.To overcome this, our project makes the use of decentralized Application approach (dApps). In our application all the user data is stored on a block which is connected to other blocks forming a chain. As the name suggests, a decentralized application does not have a centralized server.

In Ethereum Developer Resources[6],documents on introduction to blockchain and Ethereum, the way Ethereum state changes, batches of transactions added to the blockchain, Ether needed to power transactions, using JavaScript to interact with smart contracts, libraries needed to interact with smart contracts etc are available.

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state.Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features[7].

Blockchain being an immutable and append only ledger will not allow for any tampering while also being fully transparent. In this paper[8], we have implemented and tested a sample app running as a smart contract for ethereum network using E-Wallets.

Truffle takes care of managing your contract artifacts so you don't have to. Includes support for custom deployments, library linking and complex Ethereum applications.You can declare contract dependencies using Solidity's import command. Truffle will compile contracts in the correct order and ensure all dependencies are sent to the compiler[9].

Ganache 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.You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.Ganache comes in two flavors: a UI and CLI. Ganache UI is a desktop application supporting both Ethereum and Corda technology. The command-line tool, ganache-cli (formerly known as the TestRPC), is available for Ethereum development[10].

2.1 Objectives

- To provide more secure environment for chatting.
- To provide more efficient system that works even if a node in the network fails.

Chapter 3

IMPLEMENTATION

In this project, an application is developed that makes use of blockchain in a very efficient way. By eliminating the centralized approach, the safety and availability of data and communication is assured. Decentralized applications tend to make the interaction between two people more efficient and simple. The chatting process nowadays have a mediating node, while this proposed system does not have any mediating device/node.The major advantage of the system is the use of blockchain as it provides transparency in transactions. The proposed decentralized system is implemented on Ethereum Blockchain network.

3.1 System Study

3.1.1 Existing System

Over the last decade, WhatsApp, WeChat, etc, these traditional applications have taken all over the internet. There is a centralized server which stores all the information including identity to chats. Generally, these chat applications based on the following:
Centralized Management: In this system, entire correspondence goes through the company's server which can govern its rules. Messages can be blocked on a certain subject or restrictions can be applied on the certain files.
Centralized Architecture: In this architecture, there is only single server which is maintaining all the services. It is allows to block access to a certain service for the whole country whereas, the problems on the management servers may lead to inadequacy of the service for all or a significant part of users.
Confidentiality: Confidentiality of a user can be compromised on the request of government.
Single Point of Failure (SPF): If a single node fails then whole application can be compromised.
The above encouraged to build an application where, all the features like Decentralized storage, Censorship resistance, Data security and Data immutability can be implemented.

3.1.2 Proposed System

The most important features of the proposed system are:
1. Data immutability: The inability to make adjustments to the data after they are recorded.
3. Censorship resistance: The average user's ability to make immutable and trustless transactions on a blockchain network without permission from a third party.
4. Decentralized storage: It is a model of network online storage where data is stored on multiple nodes (computer), which is hosted by the participants in the network.
5. Data Security: It means protecting digital privacy measures that are applied to prevent unauthorized access to the nodes.

The front-end of proposed system is generally built on ReactJS. It's an open source and component-based JavaScript library used for creating dynamic and interactive user interfaces, especially for single page application. The back-end is Ethereum network. The blocks of network contain the information about transactions. The smart contract which is written in Solidity is deployed on test network. When the user sends the message to the particular user's address. The message block will be added to the Ethereum network according to the given prescribed contract.

3.2 Prerequisties

The various technologies and environments used in this project are:

- Ethereum
- Truffle
- Ganache
- Smart Contracts
- Metamask
- Node js
- React js
- Web3 js

3.2.1 Ethereum

Ethereum is a platform for decentralized applications. Ether is the cryptocurrency used in the platform. Smart contracts allow participants to transact with each other without a trusted central authority. Transaction records are immutable, verifiable, and securely distributed across the network, giving participants full ownership and visibility into transaction data. Transactions are sent from and received by user-created Ethereum accounts. Ethereum offers an extremely flexible platform on which to build decentralized applications using the native Solidity scripting language.

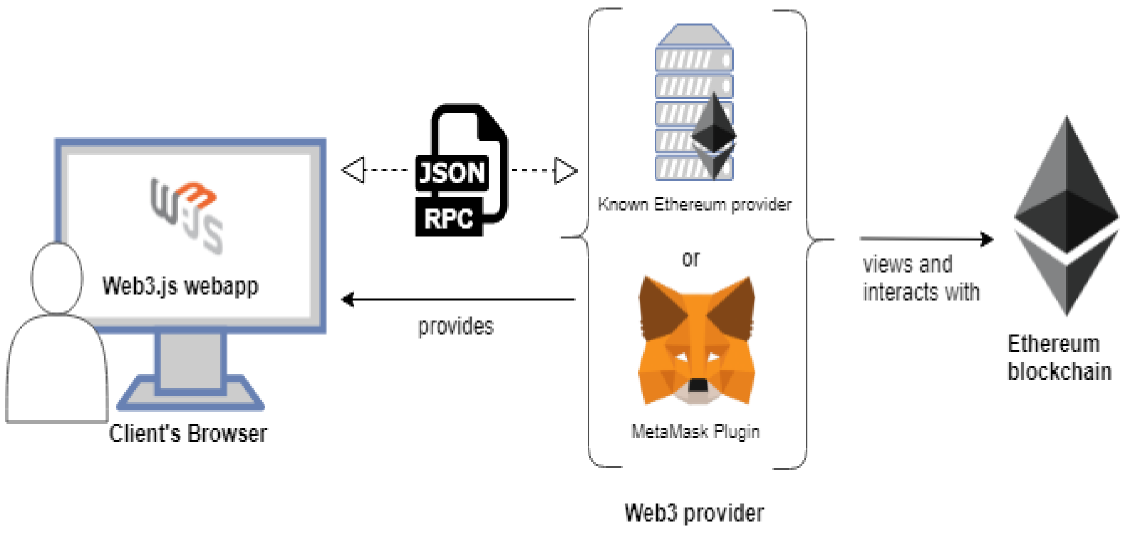


Fig 3.1: Ethereum blockchain network

In the above fig3.1 webapp provides the client side application which can be viewed, to view this needs some information this data is fetched by RPC(Re mote Procedure Call)/JSON. Metamask is the cryptocurrency wallet used to communicate with ethereum network to make transactions.

3.2.2 Truffle

Truffle is a development environment and testing framework. With Truffle, compiling and deploying Smart Contracts, injecting them into web apps, and also develop front-end for the application.Today, Truffle is one of the most widely used integrated development environment (IDEs) for Ethereum Blockchain. Truffle is the first thing that is installed to start the project. First created a project folder and truffle framework is installed using the Command Prompt. Truffle provides lots of supporting file for smooth running of the application.

3.2.3 Ganache

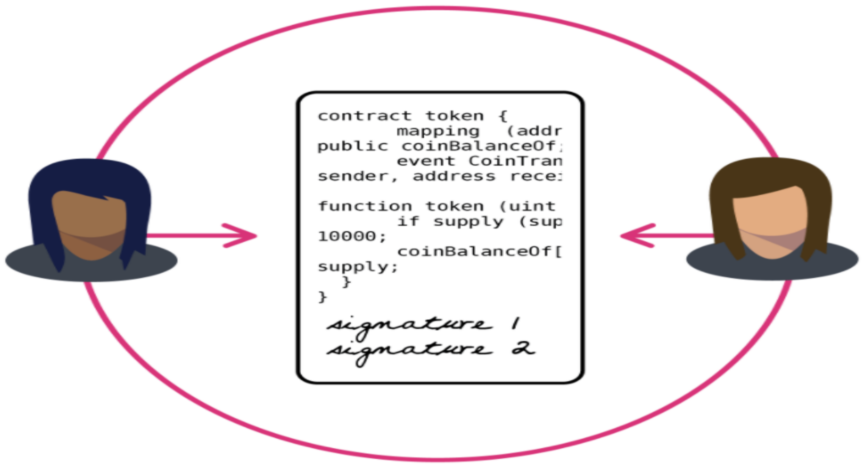
Ganache is used as the personal blockchain for Ethereum development. Ganache can be used across the entire development cycle;enabling to develop, deploy, and test the app in a safe and deterministic environment. This software can be downloaded from the website and can create new work space with 10 virtual nodes with 100 Ether for each node ,can also rename the work space and save it. Ganache is a test network to create blockchain network. The blocks created can be seen every time when a new one gets added. The transaction data are saved in the ganache in a hashed format which is not readable by others.

3.2.4 Smart Contracts

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. All of the code on the blockchain is contained in smart contracts, which are programs that run on the blockchain. They are the building blocks of blockchain application. Smart contracts are responsible for fetching all of the tasks from the blockchain, adding new tasks, and completing tasks.

Smart contracts are written in a programming language called Solidity, which looks a lot like JavaScript. It is necessary to code the smart contract according to the program. After that the code can be compiled with truffle framework. Solc is the compiler used to compile the .sol file. The compiler will also produce the Application Binary Interface (ABI) which is needed in order to understand the contract and call the contract's functions.

A Javascript client library will read the ABI in order to call the smart contract in the web app's interface. It is needed to deploy the smart contract in order for it to be available to users in the Ethereum network. To deploy a smart contract, merely send an Ethereum transaction containing the code of the compiled smart contract without specifying any recipients, a change in the ganache can be seen after deploying the smart contract. Once the contract is deployed it cannot be revoked by any one. It will be permanently saved in the blockchain network.



```
contract token {
        mapping  (addr
public coinBalanceOf;
        event CoinTran
sender, address rece:

function token (uint
        if supply (sup
10000;
        coinBalanceOf[
supply;
   }
}
```

Fig 3.2: Smart Contract

In fig 3.2 a model of smart contracts which is binding agreements contained in a computer program (codes) can be seen. Smart contracts are written in Ethereum's main language, Solidity.

3.2.5 Metamask

MetaMask is a free browser extension that allows users to store and swap crypto, interact with Ethereum, and host dApps.With its simplified user experience (UX) and approachable design, MetaMask has become a favored wallet tool for many users in the Ethereum ecosystem. As a wallet that provides storage, swaps, and access to decentralized apps (dApps), Metamask is a multifunctional, robust, and user-friendly portal to Web3. MetaMask has three primary uses: storage, swaps, and dApp access. To install Metamask, search for the Metamask Chrome plugin in the Google Chrome web store. Once installed, make sure that it is checked in the list of extensions. A fox icon can be seen in the top right hand side of Chrome browser when it's installed. Metamask will also allow to manage the personal account when connected to the blockchain, as well as manage the Ether funds that is needed to pay for transactions.

3.2.6 Node js

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project! Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language. Now the private blockchain running, need to configure the environment for developing smart contracts. The first dependency needed is Node Package Manager, or NPM, which comes with Node.js. It can see if you have node already installed by going to the terminal and typing 'node -v'.

3.2.7 React js

React makes it easy to create interactive UIs.First design simple views for each state in the application, and React will efficiently update and render just the right components when the data changes.Declarative views make the code more predictable and easier to debug. It is difficult to make assumptions about the rest of the technology stack, so developing new features in React without rewriting existing code can be made.React can also render on the server using Node and power mobile apps using React Native.

3.2.8 Web3 js

Web3.js is a collection of libraries which allows to interact with a local or remote ethereum node, using a HTTP or IPC connection. The web3 JavaScript library interacts with the Ethereum blockchain. It can retrieve user accounts, send transactions, interact with smart contracts, and more.

3.3 Working

The proposed chat application is built using Solidity, Metamask, ganache and built on top of Ethereum network, so the core of this application is smart contract which helps in transferring the messages between the nodes. Ganache and Truffle to handle the Ethereum smart-contract transactions. The web interface for the chat application was built using NodeJS, JavaScript, HTML, CSS and Bootstrap.

Before the development starts, all needed dependencies and environements are installed. Then configuration of metamask is made to interact with blockchain network. Smart contracts are created and the deployed. To migrate contracts, truffle and ganache is used.In order to communicate between the blockchain and the browser, JavaScript library named web3.js is used. web3 is an API which maps high level commands to low level RPC instructions sent to the Ethereum blockchain.By connecting web3 to an instance of Ganache server it is possible to access the data of the blockchain via API methods. Sending messages via the blockchain will be done using Solidity events.

3.4 Modules

3.4.1 Environment Setup

Before diving into blockchain and smart contract development, first install all the needed dependencies and environements. Here is a list of the dependencies needed to install before starting coding:

- node.js
- react.js
- web3.js
- metamask
- truffle suite
- ganache

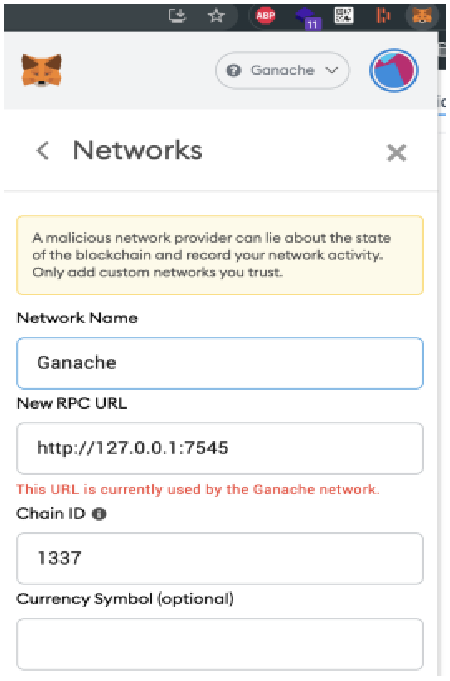
Then install the dependencies from package.json- npm install and npm run start

3.4.2 Deploy the first smart contract

A large amount of information represented in graphic form is easier to understand and analyze.

(a) Connect metamask to the browser

First of all, have to configure metamask. Metamask is a browser plugin used to configure personal ledger wallets used to store tokens that will be used when interacting with the future blockchain. After installing the plugin on the store, create a new network by clicking custom RPC at the top right of the extension.Then connect the plugin with the corresponding information used in the project. The below given figure shows how to setup a metamask plugin.



A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.

Network Name

Ganache

New RPC URL

http://127.0.0.1:7545

This URL is currently used by the Ganache network.

Chain ID

1337

Currency Symbol (optional)

Fig 3.3: Creating a new metamask network

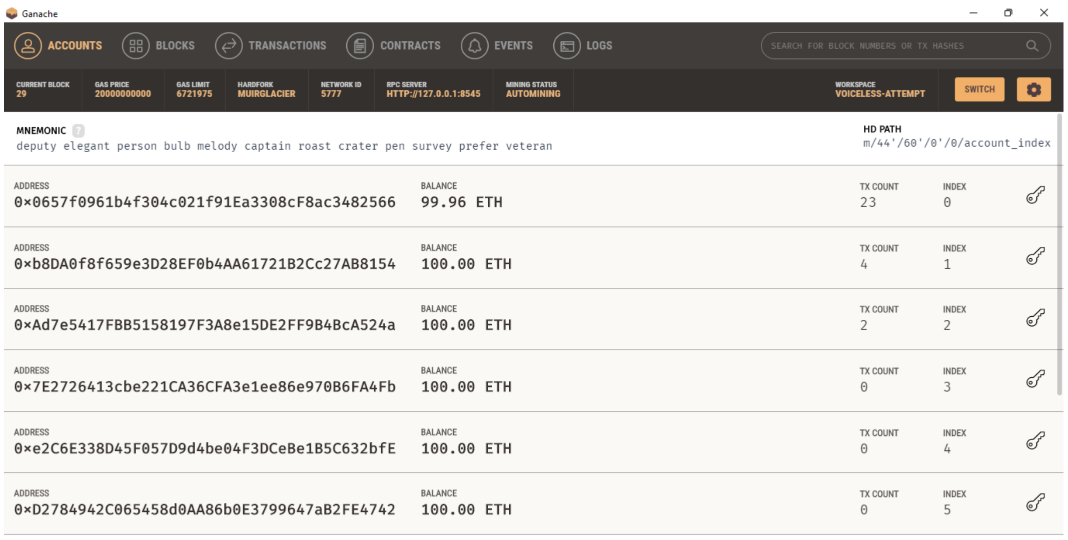
(b) Create the smart contract

Create a new file named ChatApp.sol and write very small and simple contract. This smart contract will be used to verify whether successfuly connected to ganache. Then under migrations/ create a file named contracts.js where the blockchain is to deploy the smart contract.

(c) Deploy the smart contract

To migrate the contracts, needs a truffle and ganache. Ganache is a software used to fire a personal Ethereum blockchain on a personal machine.It is needed to run the ganache app in order to start the server on which the communication with the blockchain is done. Now using the cli of truffle, can run a migration (deploy all the available smart contracts to the Ethereum blockchain). This migration can be done using the command ”truffle migrate”.Now that the contract is deployed and is possible to interact with it using commands in the truffle interpreter.

When looking to ganache interface, it is seen that the first wallet address has decreased by 0.01 ETH. This is because every transaction done on a Ethereum blockchain isn't free,and has to pay. This price that is payed to make transactions is called gas. This gas is fixed by the blockchain and is applied to all participants using the blockchain.The below figure shows the accounts available in ganache.



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
29	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:8545	AUTOMINING	VOICELESS-ATTEMPT

MNEMONIC
deputy elegant person bulb melody captain roast crater pen survey prefer veteran

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

ADDRESS	BALANCE	TX COUNT	INDEX
0×0657f0961b4f304c021f91Ea3308cF8ac3482566	99.96 ETH	23	0
0×b8DA0f8f659e3D28EF0b4AA61721B2Cc27AB8154	100.00 ETH	4	1
0×Ad7e5417FBB5158197F3A8e15DE2FF9B4BcA524a	100.00 ETH	2	2
0×7E2726413cbe221CA36CFA3e1ee86e970B6FA4Fb	100.00 ETH	0	3
0×e2C6E338D45F057D9d4be04F3DCeBe1B5C632bfE	100.00 ETH	0	4
0×D2784942C065458d0AA86b0E3799647aB2FE4742	100.00 ETH	0	5

Fig 3.4: View from Ganache

3.4.3 Send Messages

At this step, the local blockchain network is setup, now moving to the code of blockchain messaging app! These are the features implemented by the web app:
• connect to all the available wallet addresses available in Ganache
• send messages between these addresses
• monitor the state of the blockchain in real time when the transactions are executed
• bonus: send ethereum between the addresses

(a) Connect browser to Ganache wallets

In order to communicate between the blockchain and the browser, a JavaScript library named web3.js is used. In this project,UI framework react.js is used because it is easy to use and allows reactive programming. Basically, web3 is an API which maps high level commands to low level RPC instructions sent to the Ethereum blockchain, just have to connect web3 to an instance of Ganache server and then can access the data of the blockchain via API methods.

Now that the blockchain is connected, it is possible to fetch data from the blockchain. In the following function,fetching all the available account addresses in Ganache and store them in the state of the react component. Note that we store the first account of the list of accounts as the specific account of the current user on the page (the default address used to send messages later) and also fetch the instance of the Chat smart contract that is created previously. This instance is fetched using the abi json file of the contract: smart contract serialized as json file when compiling and deploying the smart contract using truffle. This json contains many information in order to find the contract when parsing it. The contract instance will be used later. Note that these two functions are triggered at the loading of the react component (page).

(b) Address selection

To send messages,should open two different pages in a browser, select two distinct addresses and send text messages between them. The first thing to do is to create a UI component so the user can select an address to send the messages with and use a select html input for this address selection feature.

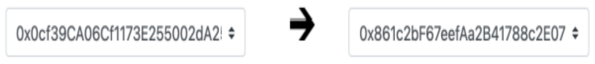


Fig 3.5: Address selection: sender and recipient

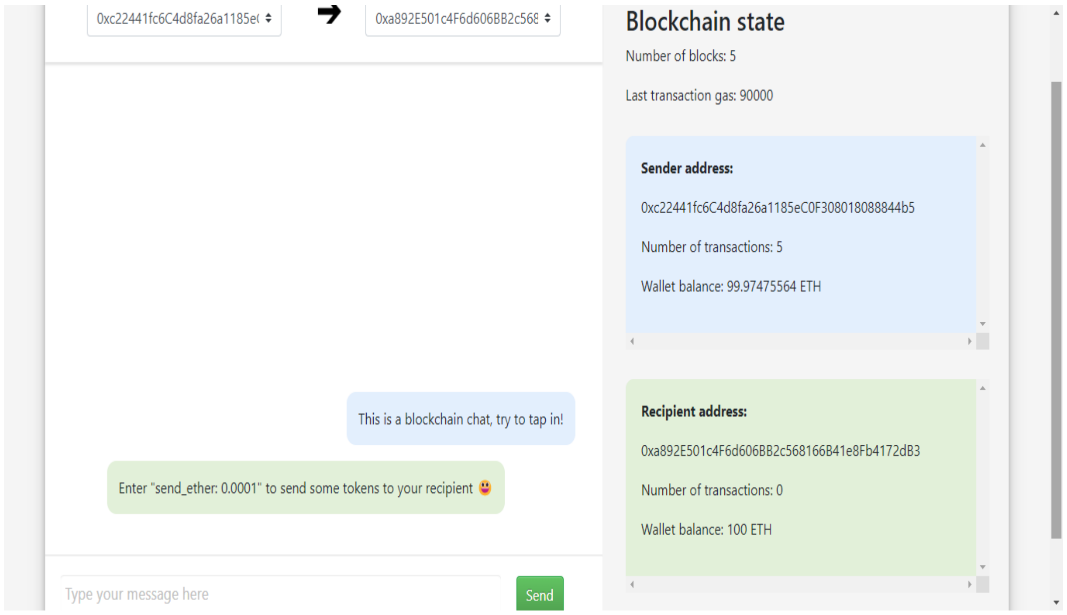
The above fig shows selection of both sender and receiver address.Each selection of a new address triggers a function which save the selected address as the current address of the user.

(c) Send messages

Sending messages via the blockchain will be done using Solidity events. Events work the same way as with software architectures like MVVM or Block. A client is listening for incoming data after subscribing to a type of event. Then an event can be sent and all the clients will be notified. This is a perfect architecture for messaging. Note that each event uses some gas on the blockchain so it has a price. In general, everything that uses power on a blockchain is not free, have to pay that power using tokens (ETH here).The message is sent using the ChatApp contract that is saved in state previously and can access the methods of the contract and then use this method:sendMsg(to address, message)

(d) Monitor real-time messaging

Using web3, it is possible to fetch a vast quantity of information concerning the blockchain. In the following examples,if fetched the ETH wallet of the two participants of the conversation, their addresses and number of transactions, but also the number of the last block in the blockchain and the gas used for the last transaction. Note that the change in the gas parameter in send() function, the last transaction gas will be the same as the one used in the send() function.The below figure shows the UI of the messaging app.



Blockchain state

Number of blocks: 5

Last transaction gas: 90000

Sender address:

0xc22441fc6C4d8fa26a1185eC0F308018088844b5

Number of transactions: 5

Wallet balance: 99.97475564 ETH

Recipient address:

0xa892E501c4F6d606BB2c568166B41e8Fb4172dB3

Number of transactions: 0

Wallet balance: 100 ETH

This is a blockchain chat, try to tap in!

Enter "send_ether: 0.0001" to send some tokens to your recipient 😛

Fig 3.6: Messaging-side

(e) Send ether

To better understand the transations in the Ethereum blockchain, it is interesting to check how the the real ether transactions between wallet addresses work.

Input format for ether sending:
The following syntax is used to send a transaction of n ether ”send ether : n”, where n is a floating point number such as 0.0002, 3, 0.123 representing the number of ether to send. If the number of ether sent if larger than the displayed wallet amount, an error will appear.

Transaction from the smart contract:
Sending ether tokens from an address to another is trivial using Solidity. The function has to be ”payable” because we are looking to send tokens. The amount of ether to transfer is passed by msg.value.

Chapter 4

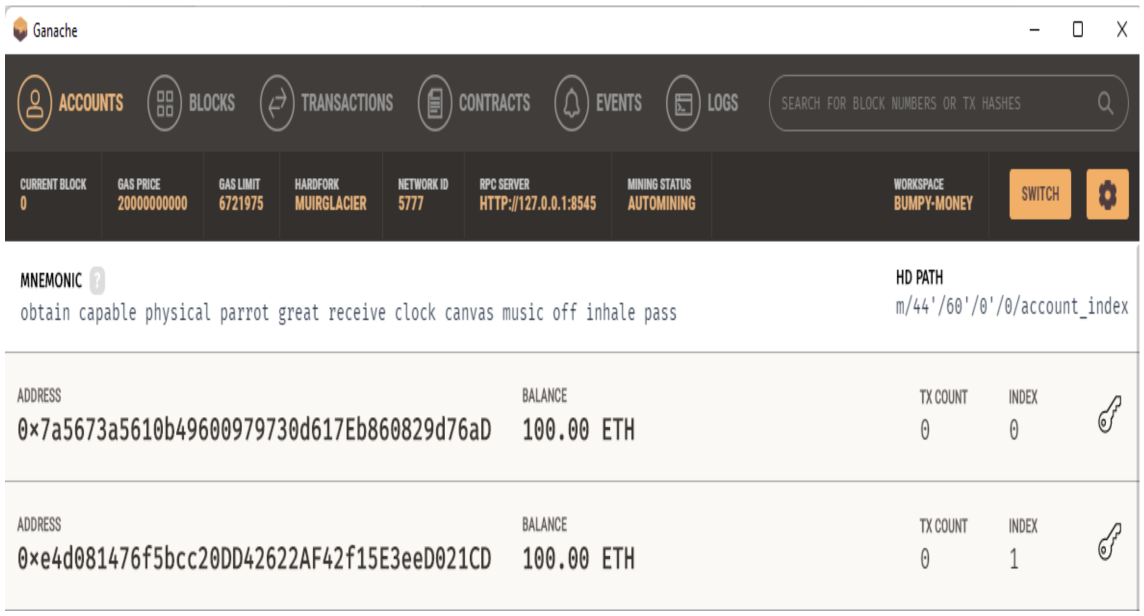
RESULT ANALYSIS

The result of the proposed project 'Decentralized Chat System' is to create a secure and reliable messaging application that overcomes the drawbacks of traditional messaging applications. As the name suggests, a decentralized application does not have a centralized server, control is distributed between participants in the system. In the application all the user data is stored on a block which is connected to other blocks forming a chain.

The decentralized application is implemented on Ethereum blockchain network. Ethereum is a decentralized blockchain platform that securely executes and verifies application code, called smart contracts. Smart contracts allow participants to transact with each other without a trusted central authority.

Ganache is used for setting up a personal Ethereum Blockchain for testing the Solidity contracts. MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension, which can then be used to interact with decentralized applications.

The below figure shows an example of a user sending 42 ether between two selected addresses. Almost instantly after sending the transaction, the balances of wallets are updates, another block is added, and transactions are incremented.To better understand the transations in the Ethereum blockchain, lets check how the real ether transactions between wallet addresses work.The fig 4.1 shows two address with same ether balance.The second fig 4.2 shows the transaction where 42 ether is send to one of the address.In the third fig 4.3, we can seee the wallet balance after the transaction.



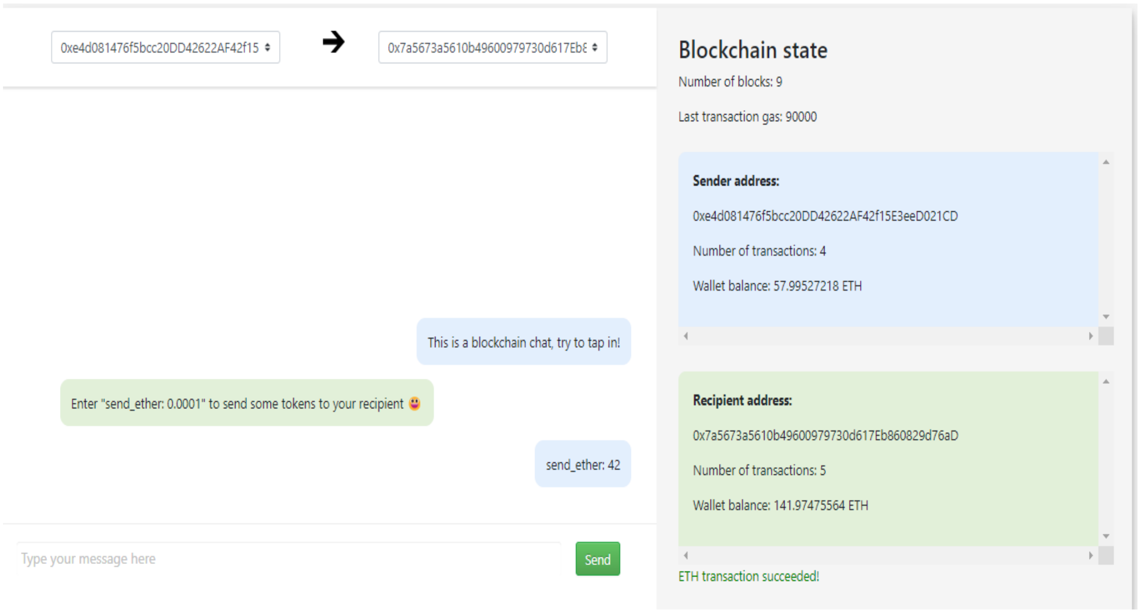
CURRENT BLOCK 0
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MUIRGLACIER
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:8545
MINING STATUS AUTOMINING
WORKSPACE BUMPY-MONEY

MNEMONIC
obtain capable physical parrot great receive clock canvas music off inhale pass

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

ADDRESS	BALANCE	TX COUNT	INDEX
0x7a5673a5610b49600979730d617Eb860829d76aD	100.00 ETH	0	0
0xe4d081476f5bcc20DD42622AF42f15E3eeD021CD	100.00 ETH	0	1

Fig 4.1: Wallets before transaction of 42 ether


0xe4d081476f5bcc20DD42622AF42f15
0x7a5673a5610b49600979730d617EbE
Send

Blockchain state

Number of blocks: 9

Last transaction gas: 90000

Sender address:

0xe4d081476f5bcc20DD42622AF42f15E3eeD021CD

Number of transactions: 4

Wallet balance: 57.99527218 ETH

Recipient address:

0x7a5673a5610b49600979730d617Eb860829d76aD

Number of transactions: 5

Wallet balance: 141.97475564 ETH

ETH transaction succeeded!

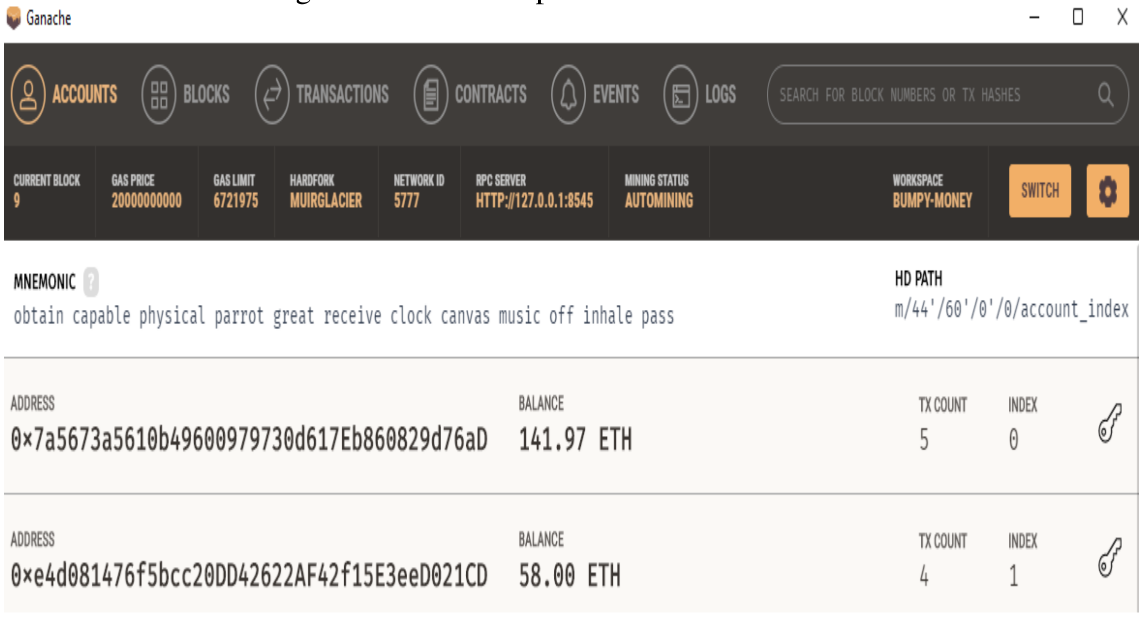
This is a blockchain chat, try to tap in!

Enter "send_ether: 0.0001" to send some tokens to your recipient 😜

send_ether: 42

Type your message here

Fig 4.2: Transaction processed



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
9	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:8545	AUTOMINING	BUMPY-MONEY

MNEMONIC

obtain capable physical parrot great receive clock canvas music off inhale pass

HD PATH

m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0x7a5673a5610b49600979730d617Eb860829d76aD	141.97 ETH	5	0
0xe4d081476f5bcc20DD42622AF42f15E3eeD021CD	58.00 ETH	4	1

Fig 4.3: Wallets after transaction of 42 ether

Chapter 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

Sending messages always been a security concern over insecure channels. Although, there are numbers of techniques to encrypt the messages but still, there are possibilities to attack on the messages like Eavesdropping, MITM, EFAIL, etc. Many countries, like China and South Korea, Citizens are being tracked by their government (e.g., what are they chatting, sharing, etc). Moreover, the traditional application manages their data on centralized database which is also a concern. This blockchain based chat app, ensures decentralization, immutability,and data security. The data which is send by the users will be directly added to the blockchain and creates the global copy of data in each node. Only the legitimate users can access that data by their private key on the blockchain. It eliminates the need for trusted intermediaries. The system is entirely decentralized and allows users to exchange message securely.

In this project, an approach of decentralized application is used. All the user data is stored on a block which is connected to other blocks forming a chain, tampering the data which is stored on the blockchain is quite impossible. If malicious user tries to make changes to the information in block then, he/she will have to make changes to all the copies of that block on whole blockchain network and that can be quite impossible. Though blocks are on all nodes, they cannot access the information in it, only the person for whom the information is concerned, they can only access.

5.2 Future Scope

Some more features can be added in future work like:

- To implement using RSA encryption for messages
- To implement using IPFS for file sharing
- To optimize gas consumption in smart contract

Chapter 6

APPENDIX

6.1 Source Code

6.1.1 Chat.js

```
import Web3 from 'web3';
import React, { Component } from 'react';
import ChatApp from '../abis/ChatApp.json'

class Chat extends Component {

async componentWillMount()
await this.loadWeb3()
await this.loadBlockchainData()
await this.listenToMessages()
await this.listenToEther()
await this.listenToAskEther()
await this.listenToFetchAllMsg()
await this.fetchAllMsg()
await this.updateUIData()
}

constructor(props) {
super(props)
let chats = [
{ msg: ”This is a blockchain chat, try to tap in!”,
response: true
},
{
msg: ”Enter ”send_ether: 0.0001” to send some tokens to your recipient ”,
response: false
}
]
this.state = {
fixedChats: chats,
chats: [],
inputValue: ”,
accounts: [],
account: ”,
nbBlocks: 0,
otherAccount: ”,
accountNbTransactions: 0,
otherAccountNbTransactions: 0,
accountBalance: 0,
otherAccountBalance: 0,
lastGas: 0,
blockHash: ”,
didATransaction: false,
isLastTransactionSuccess: false,
didARequest: false,
accountRequesting: ”,
accountRequested: ”,
valueRequested: 0,
}
}

# ——- init ——
async loadWeb3() {
if (window.ethereum) {

#automatic connection with metamask
window.web3 = new
Web3(Web3.providers.WebsocketProvider(”ws://localhost:8545”))
await window.ethereum.enable()
}
else if (window.web3) {
window.web3 = new Web3(window.web3.currentProvider)
}
else {
window.alert(’Non-Ethereum browser detected. You should consider
trying MetaMask!’)
}
}

async loadBlockchainData() {
const web3 = window.web3
const accounts = await web3.eth.getAccounts()
this.setState({
accounts: accounts,
account: accounts[0],
otherAccount: accounts[1]
})
console.log(accounts)

const ethBalance = await web3.eth.getBalance(this.state.account)
this.setState( ethBalance })

# Load smart contract
const networkId = await web3.eth.net.getId()
const chatAppData = ChatApp.networks[networkId]
const abi = ChatApp.abi
if(chatAppData) {
const chatContract = new web3.eth.Contract(abi, chatAppData.address)
this.setState({ chatContract: chatContract })
}
else {
window.alert('Chat contract not deployed to detected network.')
}
}

# ——- listeners ——
async listenToMessages() {
var binded = this.didReceiveMessageBinded.bind(this)
this.state.chatContract.events.messageSentEvent({})
.on('data', binded)
.on('error', console.error);
```

```
}

async listenToEther() {
var binded = this.didReceiveEtherBinded.bind(this)
this.state.chatContract.events.etherSentEvent({})
.on('data', binded)
.on('error', console.error);
}

async listenToAskEther() {
var binded = this.didReceiveAskEtherBinded.bind
this.state.chatContract.events.etherAskEvent({})
.on('data', binded)
.on('error', console.error);
}

async listenToFetchAllMsg() {
var binded = this.didReceiveAllMsgBinded.bind(this)
this.state.chatContract.events.messagesFetchedEvent({})
.on('data', binded)
.on('error', console.error);
}

# ——- handlers ——
async didReceiveMessageBinded(event){
const message = event.returnValues.message
if (event.returnValues.from === this.state.account)
this.didReceiveMessage(message, true)
}
```

```
if (event.returnValues.to === this.state.account){
this.didReceiveMessage(message, false)
}
this.setState({
didATransaction: false,
didARequest: false,
})
await this.updateUIData()
}

async didReceiveEtherBinded(event) {
this.setState({
didATransaction: true,
didARequest: false,
isLastTransactionSuccess: event.returnValues.success
})
await this.wait()
await this.updateUIData()
}

async didReceiveAskEtherBinded(event){
if (this.state.account === event.returnValues.to) {
let value_as_wei = window.web3.utils.fromWei( event.returnValues.value,
"ether")

this.setState({
didATransaction: false,
didARequest: true,
accountRequesting: event.returnValues.from,
accountRequested: event.returnValues.to,
valueRequested: value_as_wei,
})
await this.updateUIData()
}
}

async didReceiveAllMsgBinded(event){
let allMsg = []

event.returnValues.messages.forEach((message) = {
allMsg.push({
msg: message['message'],
response: message['from'] === this.state.account
})
})
if (allMsg.length === 0)
allMsg = this.state.fixedChats

this.setState(
chats: allMsg
})
await this.updateUIData()
}

async didReceiveMessage(message, isResponse) {
let chats = this.state.chats
chats.push(
{
msg: message,
response: isResponse
}
)
this.setState(
chats: chats,
inputValue: ”
})
}

async didSendMessage(message) {
this.state.chatContract.methods.sendMsg(this.state.otherAccount,
message)
.send({ from: this.state.account, gas: 1500000 })
await this.sendEtherIfAsked()
await this.askEtherIfAsked()
}

async sendEtherIfAsked() {
let splitted = this.state.inputValue.split(’:’)
if (splitted.length !== 2)
return false

if (splitted[0] == ”send_ether”this.isNumeric(splitted[1])){
letasWei = parseFloat(splitted[1]) ∗ 1e18
this.state.chatContract.methods.sendEther(this.state.otherAccount).send({from :
this.state.account,
value : asWei
})
returntrue
}
```

```
returnfalse
}

async askEtherIfAsked() {
let splitted = this.state.inputValue.split(’:’)
if (splitted.length !== 2)
return false

if (splitted[0] == ”ask_ether” && this.isNumeric(splitted[1])) {
var asWei = (parseFloat(splitted[1]) * 1e18).toString()
this.state.chatContract.methods.askEther(this.state.otherAccount,
asWei).send({ from: this.state.account })
return true
}
return false
}

async fetchAllMsg() {
await
this.state.chatContract.methods.getAllMsg(this.state.otherAccount).send({
from: this.state.account })
}
```

6.1.2 ChatApp.sol

```
contract ChatApp {

mapping (address =  mapping (address =  Message[])) public messages;

struct Message{
string message;
address from;
}

event messageSentEvent(address indexed from, address indexed to, string
message);
newlinenewline event etherSentEvent(address indexed from, address
indexed to, bool
success);
event etherAskEvent(address indexed from, address indexed to, string
value);
event messagesFetchedEvent(address indexed from, address indexed to,
Message[] messages);

function sendMsg(address to, string memory message) public {
messages[msg.sender][to].push(Message(message, msg.sender));
messages[to][msg.sender].push(Message(message, msg.sender));
emit messageSentEvent(msg.sender, to, message);
}
function sendEther(address payable to) public payable {
bool sent = to.send(msg.value);
emit etherSentEvent(msg.sender, to, sent);
require(sent, ”Failed to send Ether”);
}
function askEther(address to, string memory value) public {
emit etherAskEvent(msg.sender, to, value);


function getAllMsg(address to) public {
if (messages[msg.sender][to].length == 0) {
emit messagesFetchedEvent(msg.sender, to, messages[to][msg.sender]);
}
else {
emit messagesFetchedEvent(msg.sender, to, messages[msg.sender][to]);
}
}
}
```

6.1.3 Migrations.sol

```
contract Migrations
address public owner;
uint public last_completed_migration;

constructor() public {
owner = msg.sender;
}

modifier restricted() {
if (msg.sender == owner) ;
}

function setCompleted(uint completed) public restricted {
last_completed_migration = completed;
}

function upgrade(address new_address) public restricted {
Migrations upgraded = Migrations(new_address);
upgraded.setCompleted(last_completed_migration);
}
}
```

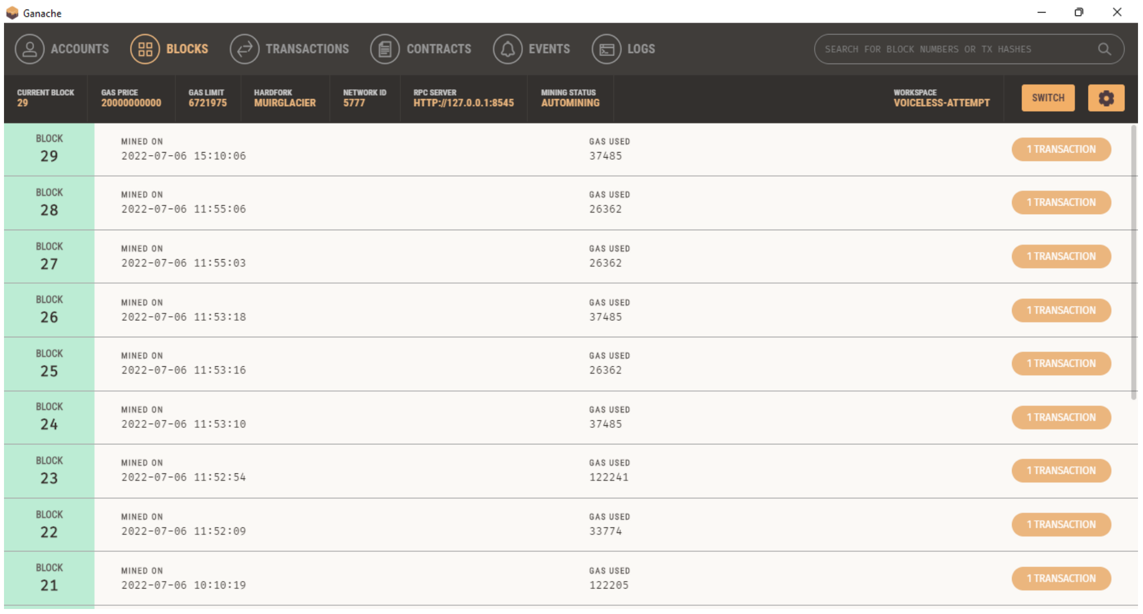
6.1.4 truffle-config.js

```
module.exports = {
networks: {
development: {
host: ”127.0.0.1”,
port: 8545,
network_id: ”*” # Match any network id
},
},
contracts_directory: ’./src/contracts/’,
contracts_build_directory: ’./src/abis/’,
compilers: {
solc: {
optimizer: {
enabled: true,
runs: 200
}
}
}
}
```

6.2 Screenshots

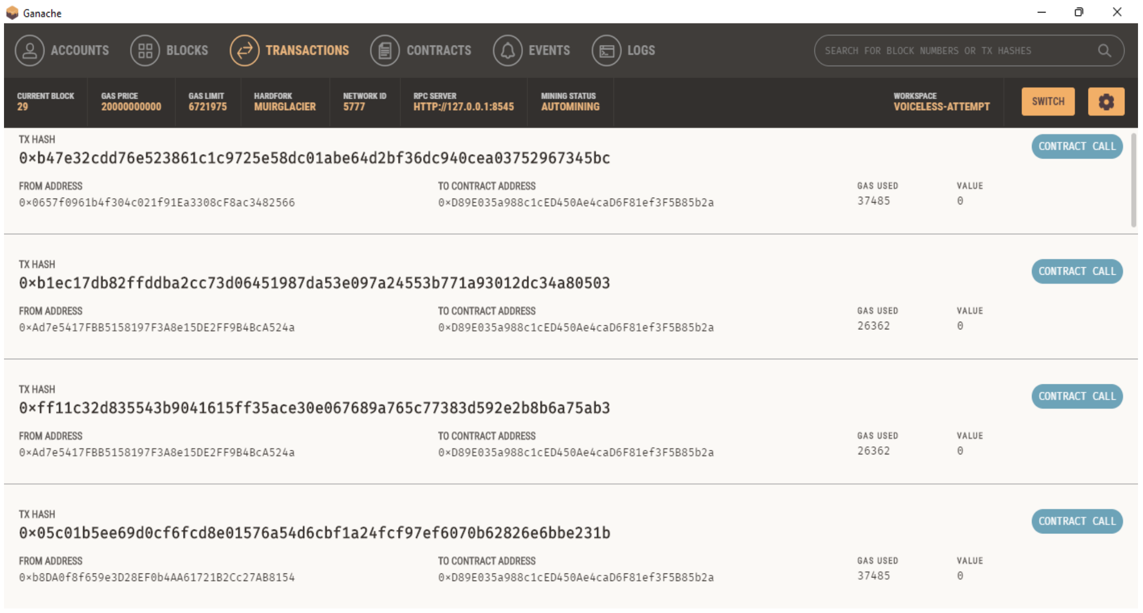
Here I'm adding some sample screenshots of decentralized chat system which includes,

- Blocks in Ganache
- Transactions in Ganache
- Address selection: sender and recipient
- Results of the messaging part
- Real time monitoring information of the chat system



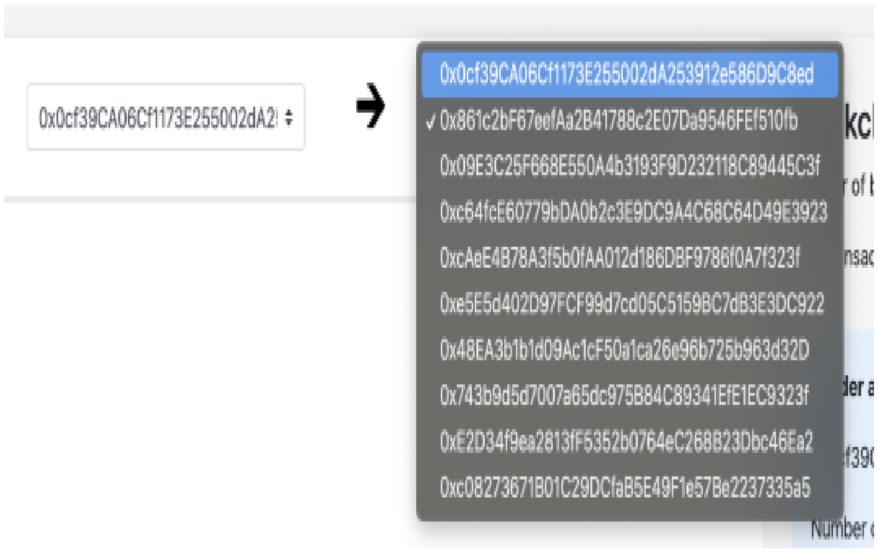
```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK      NETWORK ID  RPC SERVER              MINING STATUS  WORKSPACE
29             20000000000  6721975    MUIRGLACIER   5777        HTTP://127.0.0.1:8545   AUTOMINING     VOICELESS-ATTEMPT

BLOCK 29   MINED ON 2022-07-06 15:10:06   GAS USED 37485    1 TRANSACTION
BLOCK 28   MINED ON 2022-07-06 11:55:06   GAS USED 26362    1 TRANSACTION
BLOCK 27   MINED ON 2022-07-06 11:55:03   GAS USED 26362    1 TRANSACTION
BLOCK 26   MINED ON 2022-07-06 11:53:18   GAS USED 37485    1 TRANSACTION
BLOCK 25   MINED ON 2022-07-06 11:53:16   GAS USED 26362    1 TRANSACTION
BLOCK 24   MINED ON 2022-07-06 11:53:10   GAS USED 37485    1 TRANSACTION
BLOCK 23   MINED ON 2022-07-06 11:52:54   GAS USED 122241   1 TRANSACTION
BLOCK 22   MINED ON 2022-07-06 11:52:09   GAS USED 33774    1 TRANSACTION
BLOCK 21   MINED ON 2022-07-06 10:10:19   GAS USED 122205   1 TRANSACTION
```

Figure 6.1: Blocks in Ganache



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK      NETWORK ID  RPC SERVER              MINING STATUS  WORKSPACE
29             20000000000  6721975    MUIRGLACIER   5777        HTTP://127.0.0.1:8545   AUTOMINING     VOICELESS-ATTEMPT

TX HASH                                                                                          CONTRACT CALL
0x b47e32cdd76e523861c1c9725e58dc01abe64d2bf36dc940cea03752967345bc
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED  VALUE
0x0657f0961b4f304c021f91Ea3308cF8ac3482566    0xD89E035a988c1cED450Ae4caD6F81ef3F5B85b2a    37485     0

TX HASH                                                                                          CONTRACT CALL
0xb1ec17db82ffddba2cc73d06451987da53e097a24553b771a93012dc34a80503
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED  VALUE
0xAd7e5417FBB5158197F3A8e15DE2FF9B4BcA524a    0xD89E035a988c1cED450Ae4caD6F81ef3F5B85b2a    26362     0

TX HASH                                                                                          CONTRACT CALL
0xff11c32d835543b9041615ff35ace30e067689a765c77383d592e2b8b6a75ab3
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED  VALUE
0xAd7e5417FBB5158197F3A8e15DE2FF9B4BcA524a    0xD89E035a988c1cED450Ae4caD6F81ef3F5B85b2a    26362     0

TX HASH                                                                                          CONTRACT CALL
0x05c01b5ee69d0cf6fcd8e01576a54d6cbf1a24fcf97ef6070b62826e6bbe231b
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED  VALUE
0xb8DA0f8f659e3D28EF0b4AA61721B2Cc27AB8154    0xD89E035a988c1cED450Ae4caD6F81ef3F5B85b2a    37485     0
```

Figure 6.2: Transactions in Ganache



0x0cf39CA06Cf1173E255002dA2

0x0cf39CA06Cf1173E255002dA253912e586D9C8ed

✓ 0x861c2bF67eefAa2B41788c2E07Da9546FEf510fb

0x09E3C25F668E550A4b3193F9D232118C89445C3f

0xc64fcE60779bDA0b2c3E9DC9A4C68C64D49E3923

0xcAeE4B78A3f5b0fAA012d186DBF9786f0A7f323f

0xe5E5d402D97FCF99d7cd05C5159BC7dB3E3DC922

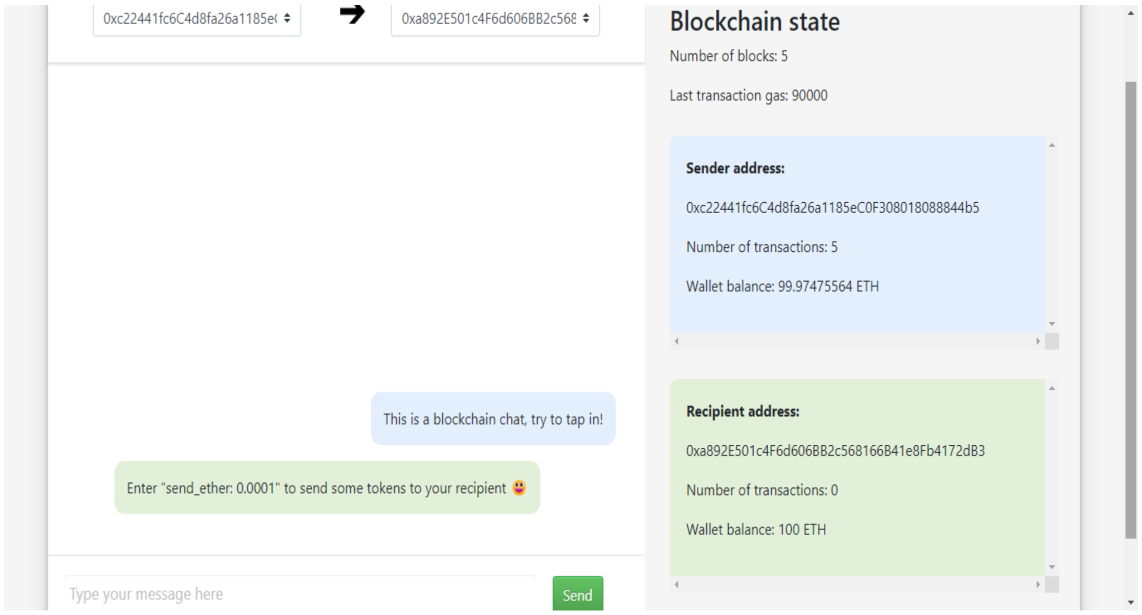
0x48EA3b1b1d09Ac1cF50a1ca26e96b725b963d32D

0x743b9d5d7007a65dc975B84C89341EfE1EC9323f

0xE2D34f9ea2813fF5352b0764eC268B23Dbc46Ea2

0xc08273671B01C29DCfaB5E49F1e57Be2237335a5

Figure 6.3: Address selection: sender and recipient



0xc22441fc6C4d8fa26a1185e

0xa892E501c4F6d606BB2c568

This is a blockchain chat, try to tap in!

Enter "send_ether: 0.0001" to send some tokens to your recipient 😛

Type your message here

Blockchain state

Number of blocks: 5

Last transaction gas: 90000

Sender address:

0xc22441fc6C4d8fa26a1185eC0F308018088844b5

Number of transactions: 5

Wallet balance: 99.97475564 ETH

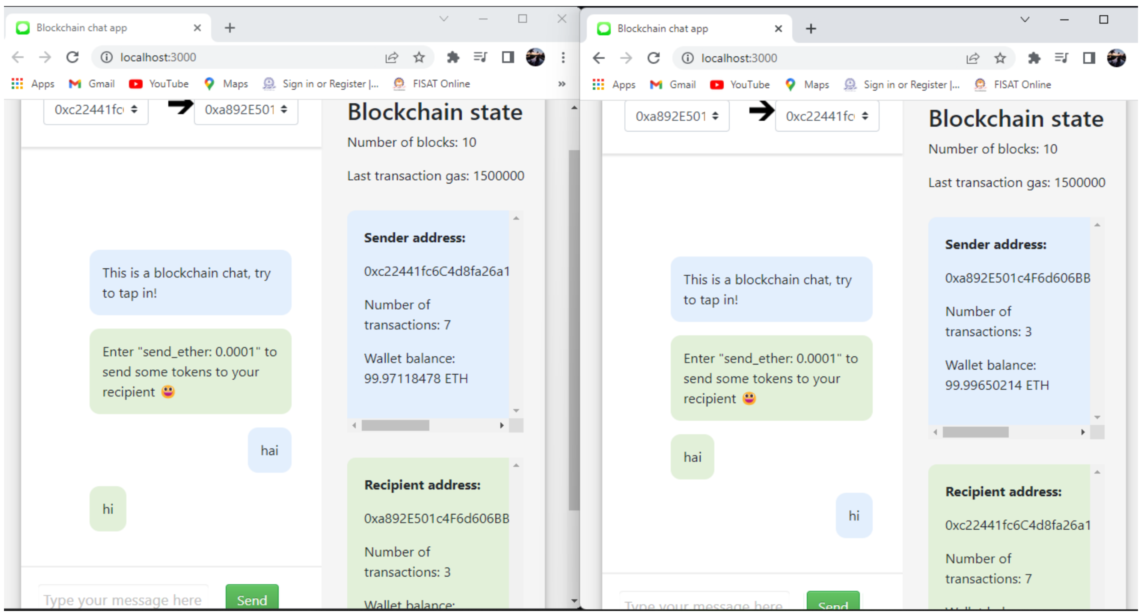
Recipient address:

0xa892E501c4F6d606BB2c568166B41e8Fb4172dB3

Number of transactions: 0

Wallet balance: 100 ETH

Figure 6.4: Results of the messaging part



0xc22441fc → 0xa892E501

This is a blockchain chat, try to tap in!

Enter "send_ether: 0.0001" to send some tokens to your recipient 😀

hai

hi

Blockchain state

Number of blocks: 10

Last transaction gas: 1500000

Sender address:

0xc22441fc6C4d8fa26a1

Number of transactions: 7

Wallet balance: 99.97118478 ETH

Recipient address:

0xa892E501c4F6d606BB

Number of transactions: 3

Wallet balance:

0xa892E501 → 0xc22441fc

This is a blockchain chat, try to tap in!

Enter "send_ether: 0.0001" to send some tokens to your recipient 😀

hai

hi

Blockchain state

Number of blocks: 10

Last transaction gas: 1500000

Sender address:

0xa892E501c4F6d606BB

Number of transactions: 3

Wallet balance: 99.99650214 ETH

Recipient address:

0xc22441fc6C4d8fa26a1

Number of transactions: 7

Figure 6.5: Real time monitoring information of the chat system

Chapter 7

REFERENCES

[1] Yaga, Dylan, et al. 2019 ”Blockchain technology overview.”

[2] Adam Hayes, 2022 ”Block chain Guide”

[3] Katerina Pace, 2022 ”ETHEREUM DEVELOPMENT”

[4] Udemy, “Become a Blockchain Developer with Ethereum and Solidity” by Sebastien Arbogast, Said Eloudrhiri

[5] Decentralized Chat Application using Blockchain Technology Abhishek P. Takale, Chaitanya V. Vaidya, Suresh S. Kolekar Rajendra Mane College Of Engineering And Technology, Ambav, India

[6] Ethereum Developer Resources: https://ethereum.org/developers/

[7] Solidity documentation: https://solidity.readthedocs.io/en/v0.6.5/

[8] Fifth International Conference on Science Technology Engineering and Mathematics (ICONSTEM) 978-1-7281-1599-3/19/ ©2019 IEEE 75 De-

centralised Applications Using Ethereum Blockchain

[9] Truffle : https://truffleframework.com

[10] Ganache: https://truffleframework.com/ganache