*A project report on*

**DECENTRALIZED CHAT APPLICATION USING BLOCKCHAIN AND CRYPTO SYSTEMS**

*Submitted in partial fulfilment for the award of the degree of*

**Bachelor of Technology**

in

**Information Technology**

*by*

**SACHINTHRA N V (18BIT0020)**



SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

April, 2022

**DECENTRALIZED CHAT APPLICATION USING BLOCKCHAIN AND CRYPTO SYSTEMS**

*Submitted in partial fulfilment for the award of the degree of*

**Bachelor of Technology**

in

**Information Technology**

*by*

**SACHINTHRA N V (18BIT0020)**



SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

April, 2022

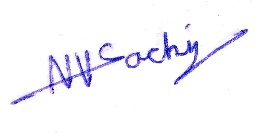
**DECLARATION**

I hereby declare that the thesis entitled “Decentralized Chat Application using Blockchain and Crypto Systems” submitted by me, for the award of the degree of Bachelor of Technology in Information Technology to Vellore Institute Technology is a record of bonafide work carried out by me under the supervision of Professor Hari Ram Vishwakarma.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

**Place: Vellore**

**Date: 22 April 2022**

**Signature of the Candidate**

**CERTIFICATE**

This is to certify that the thesis entitled “Decentralized Chat Application using Blockchain and Crypto Systems**”** submitted by Sachinthra N V (Reg. No.: 18BIT0020) School of Information Technology and Engineering Vellore Institute Technology, for the award of  
the degree of Bachelor of Technology in Information Technology is a record of bonafide work carried out by him under my supervision.

The contents of this report have not been submitted and will not be submitted  
either in part or in full, for the award of any other degree or diploma in this institute or  
any other institute or university. The Project work fulfils the requirements and regulations of  
VIT and in my opinion meets the necessary standards for submission.

**Signature of the Guide**  **Signature of the HoD**

**Internal Examiner**  **External Examiner**

**ABSTRACT**

Nowadays, blockchain systems are in great demand across many domains, which is due to its security and for its ability to maintaining decentralized record of transactions.

Traditional chat applications are centralized, that means 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. For example, WhatsApp server as a centralised server, if in case that server is destroyed or any crash in server would lead to a huge loss of user data, or they can even leak the user information stored on the server. There are many forms of threats to the server when it has a centralised server. To overcome this, my idea is to make use of decentralized Application approach that is to make a chat app backed by Blockchain.

we will see the how the traditional messaging operations works and what will happen when we implement them using blockchain’s smart contracts this will also empathise the benefits of implementing blockchain in other domains. I will be using the Solidity for implementing smart contracts.

i

**ACKNOWLEDGEMENT**

It is my pleasure to express with deep sense of gratitude to Professor HARI RAM VISHWAKARMA, Senior Professor, School of Information Technology and Engineering, Vellore Institute of Technology, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavour. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Blockchain Technology.

I would like to express my gratitude to Dr. G. Viswanathan , Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Mr. G. V. Selvam, Dr. Rambabu Kodali, Dr. S. Narayanan, Dr. V.S. Kanchana Bhaaskaran and Dr. SUMATHY S, School of Information Technology and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. USHA DEVI G. Associate Professor Grade 2, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: 22 April 2022 SACHINTHRA N V

*ii*

**CONTENTS**

**CONTENTS**  iii

**LIST OF FIGURES** ix

**CHAPTER 1**

**INTRODUCTION**

* 1. OBJECTIVE 1
  2. MOTIVATION 1

1.3 PROBLEM STATEMENT 1

1.4 SCOPE OF PROJECT 2

**CHAPTER 2**

**LITERATURE SURVEY**

2.1 LITERATURE SURVEY 3

**CHAPTER 3**

**IMPLEMENTATION**

3.1 TECHNICAL SPECIFICATION 12

3.2 DESIGN APPROACH AND DETAILS 12

3.3 MODULE DESCRIPTION 14

3.4 TESTING 20

3.5 UML DIAGRAM 27

3.6 CODE 29

*iii*

**CHAPTER 4**

**CONCLUSION & FUTURE WORK**

4.1 CONCLUSION & FUTURE WORK 49

**REFERENCE** 50

*iv*

**LIST OF FIGURES**

3.1 SYSTEM ARCHITECTURE 13

3.2 SYSTEM ARCHITECTURE OUTLINE 13

3.3 SYSTEM COMPONENT 14

3.4 CONTRACT OBJECTS 16

3.5 SMART CONTRACT CHECK USER REGISTRATION COMPONENT 17

3.6 SMART CONTRACT CLEAR INBOX 17

3.7 SMART CONTRACT REGISTER USER 17

3.8 SMART CONTRACT GET CONTRACT PROPERTIES 18

3.9 SMART CONTRACT SEND MESSAGE 18

3.10 SMART CONTRACT RECEIVE MESSAGE AND GET IMBOX SIZE 19

3.11 REMIX TESTING INTERFACE AND ACCOUNTS 20

3.12 REMIX TESTING FUNCTIONS 21

3.13 REMIX TESTING CHECK USER REGISTER AND REGISTER USER 21

3.14 REMIX TESTING GET CONTRACT PROPERTIES 22

3.15 REMIX TESTING SENDING MESSAGE 22

3.16 REMIX TESTING RECEIVE MESSAGES AND IMBOX SIZE 23

3.17 REMIX TESTING CHECK USER REGISTER AND REGISTER USER 23

3.18 FRONTEND METAMASK 24

3.19 FRONTEND INTERFACE AND USER DIRECTORY 25

3.20 FRONTEND SENDING AND RECEIVING MESSAGE 26

3.21 USE CASE DIAGRAM 27

3.22 USE CASE ACTIVITY DIAGRAM 27

3.22 CLASS DIAGRAM 28

*iv*

Chapter **1**

**Introduction**

This chapter will explain about purpose, goals and scope of the project. I will explain about the motivation, objective and scope of the project.

* 1. **MOTIVATION**

Decentralized application makes use of peer-to-peer networks, this ensures that no network failure can occur due to central node failure. Blockchain serves as an immutable ledger which allows messaging to take place in a decentralized manner. A decentralized application for communication and resource sharing is need in today’s world, where keeping data on a centralized server can be risky and costly experience. With the help of various consensus, we can implement different ways to share resources and communicate.

Together with Blockchain and Decentralized Applications, we can create a secure and reliable messaging application that overcomes the drawbacks of traditional messaging applications.

* 1. **OBJECTIVE**

To develop software that can provide all the features provided by a conventional chat application at the same time overcoming the drawbacks that it has. The resultant software will be more secure and reliable chat application. There is not a lot of decentralised chat application, even the most popular ones like WhatsApp and twitter are centralised. This really opens up the scope of such a decentralized application in a world which is pioneering towards security and reliability. Block chain has already shown the potential for transforming traditional industry. The chatting process nowadays have a mediating node, while the application proposed does not have any mediating device/node. Every person is connected by peer-to-peer network. Centralized approach to communication is vulnerable to collapse. Also, the information shared can be hacked which is stored on the central server. In a block chain, since the records are stored on a large number of participants, it is nearly impossible to tamper with the data. It also makes it equally impossible to propagate through such a large number of blocks. The proposed system looks to get past all these issues to provide a secure chatting platform.

* 1. **PROBLEM STATEMENT**

The idea is to make use of decentralized Application approach to make a chat app backed by Blockchain. I will be using the Solidity for implementing smart contracts.

1

* 1. **SCOPE OF PROJECT**

Nowadays, blockchain systems are in great demand across many domains, which is due to its security and for its ability to maintaining decentralized record of transactions. Traditional chat

applications are centralized that means 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. For example, WhatsApp server stores all the data on a central server, if in case that server is destroyed or any crash in server would lead to a huge loss of user data, or they can even leak the user information stored on the server. There are many forms of threats to the server when it has a centralised server. After completing the project, we will have a application which can be used for formal chatting purpose like sending important immutable information to other users. we will be able to understand how the traditional messaging operations works and what difference will it have when we implement them using blockchain’s smart contracts this will also empathise the benefits of implementing blockchain in other domains. Any important and secure conversations can be operated on this platform.

2

Chapter 2

**Literature Survey**

This chapter is about the background study of the topic which helps investigate a problem that no one else has addressed under this topic.

**Paper 1: Secure Messaging Platform Based on Blockchain**

Author: U. P. Ellewala; W.D.H.U Amarasena; H.V Sachini Lakmali; L.M.K Senanayaka; A.N. Senarathne

Publication: IEEE 2020

The Author(s) in this study says, several industries are also pursuing blockchainbased technology solutions to improve efficiency, streamline business processes and build trust between parties with little or no knowledge of each other. A peer-to-peer chat system on the other hand aims at replacing the centralized server with a distributed server residing at every user device. The Author’s primary objective, through this research, is to develop a chat application with more secure channels of enterprise level communication. Using new technologies such as blockchain, which operate on a decentralized model, they can surmount the drawbacks of traditional messaging applications, thereby ensuring confidentiality, integrity and availability of official data, along with advanced auditing features. This research furthermore summarizes and put forward the concept of chat application using blockchain can be more useful for the co-operate users as most of the chat applications are not be able to view chat history in present.

**Paper 2: Secure Communications Using Blockchain Technology**

Author: Peter Menegay; Jason Salyers; Griffin College

Publication: IEEE – 2018

The Authors research on the idea that the blockchain, which already provides a completely secure method of exchanging cryptocurrencies, can be extended to any transactional or communications paradigm. Email applications are described in which application-level standard protocols (e.g. SMTP, POP3, IMAP) were enhanced to optionally allow messages to be routed through a blockchain using familiar client applications such as MS Outlook or Thunderbird. Chat applications are discussed similarly, using the IRC protocol and a custom web-based protocol

3

based on Socket. IO. A MIPR application is also featured where the ability of 4multiple parties to sign documents and release funds is illustrated. The integrity of the blockchain, inherent in its distributed nature, is discussed along with enhanced software to enable quick recovery from a breach. The Authors in their work illustrates the viability of blockchain-based communications

and transactions beyond their obvious cryptocurrency application.

**Paper 3: Secure Peer-to-Peer communication based on Blockchain**

Author: Kahina Khacef, Guy Pujolle

Publication: Springer 2019

In this paper, the author discusses that the blockchain is an innovative technology that overcomes security threats and allows to decentralize sensitive operations while preserving a high level of security. It eliminates the need for trusted intermediaries. The blockchain is accessible to all network nodes and keeps track of all transactions already made. The goal of my work is to propose a secure messaging solution based on the blockchain technology. In this paper, the author explain why blockchain would make communications more secure, and they propose a model design for blockchain-based messaging maintaining the performance and security of data recorded on the blockchain, using a smart contract to verify the identities and their associated public keys, and validate the user’s certificate. The system is entirely decentralized and allows users to exchange messages securely.

**Paper 4: A Decentralized Connectivity Service for Private Social Communication Apps**

Author: Christos Aslanoglou; Michalis Konstantopoulos; Nikos Chondros; Mema

Roussopoulos

Publication: IEEE - 2020

People rely on a variety of VoIP/chat/instant messaging and other applications to ocialize, inform and collaborate with others. To use these centralized services, users must either hand over their full list of associations or have their associations tracked over time by the centralized service. This results in lack of user privacy, lock-in of users who cannot easily switch to another service, and a general stifling of application innovation. In this paper, the author proposes decoupling such 5 applications from the list of associations. He presents the Decentralized Connectivity Service (DCS), a privacy-preserving blockchain-based connectivity service for distributed social communication applications. DCS puts control of a user's interactions with others, back in the

**4**

hands of the user and allows users to establish point-to-point connections with other users, without the use of a centralized party. The author also presents the design of DCS and show how they leverage the use of a smart contract to provide monetary incentives for users to participate in the system.

**Paper 5: A blockchain based secure communication framework for community interaction**

Author: RahulSharmaa; MohammadWazida; ProsantaGope

Publication: Elsevier – 2021

The author in his research says, the existing communication frameworks for community interactions are based on the trust of intermediary institutions, for example, a central authority server. It is considered to be at the risk of single-point failure attack. Therefore, designing of a blockchain based framework for the community interaction is highly recommended. In this paper, the authors propose a blockchain based secure communication framework for community interaction. In the proposed framework, they used the mechanism of blockchain and implemented it with a tamper-proof distributed ledger. On the basis of proposed framework an android application (app) has been developed to check its usability in the real-time environment. The provided security analysis proved the security of the proposed framework against the various types of possible attacks. Lydia Chrislin Paul

**Paper 6: BlockChat: A Decentralized Messenger on the Blockchain**

Author: Andrew Zhang; Sahil Shah; Yassine Elmandjra; Mark DeCesare; Matthew Hanna

In this project, the author states that there are numerous problems with the centralized messaging services that are prevalent in the current day. These problems include a lack of interoperability between services and a lack of data ownership. The author develops a messaging framework on Blockstack, a decentralized internet, 6 because it provides the decentralized identity and storage functions that are key to their project. This app allows for group messaging between users, and it is created in a standardized way such that BlockChat users could potentially message with someone using a different decentralized messaging app on Blockstack.

**5**

**Paper 7: bChat: A Decentralized Chat Application**

Author: Sourabh, Deepanker Rawat, Karan Kapkoti, Sourabh Aggarwal, Anshul Khanna

Publication: International Research Journal of Engineering and Technology (IRJET) – 2020

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 Moreover, the traditional application manages their data on centralized database which is also a concern. In this paper, the author presents BChat, which ensures decentralization, immutability, censorship resistance and data security. The data which send by the users will 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

**Paper 8: Blockchain based secure communication application proposal:** Cryptouch

Author: Recep Ahmet Sarıtekin; Eren Karabacak; Zübeyir Durgay; Enis Karaarslan

Publication: IEEE – 2018

In this project, the author talks about the convergence which brought with the globalizing world, highly affects the network and the communication sectors. Mostly closed source and centralized systems are used for network and communication. This contradicts with the information security principles and privacy. These centralized systems cannot fully meet the concept of transparent, reliable, fast and uninterrupted communication. Distributed, decentralized and also transparent communication is possible with the blockchain technology. In this study, a communication application which is based on blockchain technology is proposed. InterPlanetary File System (IPFS) is preferred to overcome the limits of the blockchain. A prototype implementation called Cryptouch is proposed. Application features and potential benefits are discussed in this project by the author.

**6**

**Paper 9: A new type of blockchain for secure message exchange in VANET**

Author: RakeshShrestha; RojeenaBajracharya; Anish P. Shrestha; Seung Yeob Namb

Publication: KeAi – 2019

In this paper, the author explains his research, a blockchain based message exchange application VANET. In (VANET), the collection and dissemination of lifethreatening traffic event information by vehicles are of utmost importance. However, traditional VANETs face several security issues. The author proposes a new type of blockchain to resolve critical message dissemination issues in the VANET. He creates a local blockchain for real-world event message exchange which is a new type of blockchain suitable for the VANET. The author in his project presents a public blockchain that stores the node trustworthiness and message trustworthiness in a distributed ledger that is appropriate for secure message dissemination.

**Paper 10: A Decentralized Application on the Ethereum Blockchain**

Author: Ruhi Taş; Ömer Özgür Tanrıöver

Publication: IEEE – 2019

The author explains in this paper that blockchain innovation is being utilized in numerous regions, during most recent couple of years. Besides, extraordinary application openings are as yet been researched. Blockchain depends on and licenses to execute the idea of Decentralized Application (DApps). This makes the applications more transparent, distributed and flexible. The complexity of blockchain and its integration problems require expertise that differs from traditional application development approaches. Inside this unique situation, this paper gives an idea about building a DApp perhaps the most mainstream blockchain based stages called Ethereum.

**Paper 11: EtherShare: Share Information in JointCloud Environment Using Blockchain-based Smart Contracts.**

Author: Peilin Zheng; Zibin Zheng; Weili Chen; Jing Bian; Jianxun Eileen Yang

Publication: IEEE – 2019

They proposed an blockchain-based application in JointCloud computing to enhance the trust between different cloud entities. As different entities in the JointCloud environment maintain a blockchain running with the smart contracts. Then the users interact with the JointCloud entities

7

to share information or do other operations. Since the storage is kept by all the entities which are maintaining the blockchain, it is reliable enough and can be considered as permanent storage in some cases. The blockchain-based smart contracts are used to save the rights of the users. Each JointCloud entity validates the users and operates the storage through the contracts so that they can trust each other.

**Paper 12: A Blockchain based Electronic Medical Health Records Framework using Smart Contracts**

Author: Vardhini B; Shreyas N Dass; Sahana R; R. Chinnaiyan

Publication: IEEE – 2021

The main problem of the current health care is that the organizations hold multiple and fragmented medical records of patients. The Proposed System aims to solve the health care sector's current problems by hosting medical record transactions on the Blockchain to create a smart ecosystem. The goal is to provide secure access to patient data, avoiding the third party accessing it without permission. EHR Framework uses blockchain technology to securely store the records and maintain a single version of the truth. The stakeholders will have to request permission to access a patient's history and commit the transaction to the distributed ledger. A solution cantered on the blockchain, can permit large-scale availability, data confidentiality, cost-effectiveness, and belief in the information system.

**Paper 13: Decentralized Tourism Destinations Rating System Using 6AsTD Framework and Blockchain**

Author: Arif, Yunifa Miftachul, Nurhayati, Hani Harini, SriSusiki Nugroho, Supeno Mardi Hariadi, Mochamad

Publication: IEEE – 2020

This research proposed alternative methods in the tourism destinations rating system using the 6AsTD Framework as the basis of tourism destinations assessment and blockchain as a decentralized data sharing architecture. This paper has several sections to describe the research steps and explain the results. The rating assessment result of tourism destinations used to explore

8

information and data in the early stages of a tourist journeyEach rating assessment data converted to hashes and transmitted through the blocks using blockchain technology. Also they used 6AsTD as a framework for obtaining data and assessment information of destination attributes of tourism destinations conducted by tourists.

**Paper 14: Towards Secure and Decentralized Sharing of IoT Data**

Authors: Truong, Hien Thi Thu Almeida, Miguel Karame, Ghassan Soriente, Claudio

Publication: IEEE – 2019

In this they proposed a system where blockchain is used to handle transactions between parties before granting permissions, however how such transactions are done is not addressed, and only owners can change policies. Our design allows ACL updates or decryption key distributions to be autonomously done over the blockchain back-end without any data owner’s interventions. Owners only specify for data offers, trading and granting are handled by the blockchain. a framework for research data sharing that provides incentives to data owners. Proposed framework with a number of tools borrowed from system security and applied cryptography, leading to solutions with different level of trust in the off-chain components.

**Paper 15: Electronic Regulation of Data Sharing and Processing Using Smart Ledger Technologies for Supply-Chain Security**

Authors: Epiphaniou, Gregory, Prashant, Pillai, Mirko, Bottarelli, Haider, Al-Khateeb, Mohammad, Hammoudesh, and Carsten, Maple

Publication: IEEE – 2020

They implemented an underlying private permissioned BC and a distributed storage technology coupled with a novel algorithm for anchor generation used for a secure and “off-chain” data search and retrieval. The execution of our algorithm is done by the development and deployment of bespoke smart contracts directly injected in the BC network. They present a preliminary threat model against our solution with emphasis placed upon their dedicated smart contracts. Initial results of Cydon’s performance and testbed parameters. Cydon is a data management platform that employs a novel crypto-token generation engine and search mechanism which runs over a private permissioned BC. Cydon delivers an always-on audit trail of data flows, while securely distributes and retrieves data from different business entities within a private distributed network.

9

**Paper 16: A decentralized vehicle anti-theft system using Blockchain and smart contracts**

Authors: Das, Debashis Banerjee, Sourav Ghosh, Uttam Biswas, Utpal Bashir, Ali Kashif

Publication: Peer-to-Peer Networking and Applications - 2021

This paper also explains how Blockchain can be adopted for vehicle security and provides a stepwise implementation of the proposed methodology by providing test bed results and significant critical comparison analysis with other existing systems. Using the BVATS, more than one person can be authorized to drive a vehicle without hampering the vehicle data and maintaining security. Blockchain network can perceive the vehicle activities. So there is no possibility of any fraudulent activities. Blockchain can provide a significant contribution to the vehicle anti-theft system. , if one node crashes then the other node has the backup, as each node connected in the blockchain network has the same copy of data. Their system has the encryption method to access smartphone applications. Thus, personal information cannot be accessed by attackers using malware

**Paper 17: Health Record Management through Blockchain Technology**

Authors: V M Harshini; Shreevani Danai; H R Usha; Manjunath R Kounte

Publication: IEEE - 2019

They developed as a contract by the name healthcare which consists of 2 nodes that is equivalent to two people and can be assumed as hospital admin. A structure is created and named as a record which comprises of patient’s Details highlights on the patient-driven model of record maintenance using Blockchain technology where smart contracts can be incorporated in future days making it more potential in data exchange. Finding its huge scope, hoping that more researches will be carried out and practically implemented. Blockchain being a decentralised and distributed ledger it can also impact on billing, record sharing, medical research, identify thefts and financial data crimes in days to come. Implementation of smart contracts in health care can simplify things even better. Where invoking, record creation and validation will be done on Blockchain.

**Paper 18: Implementing decentralized auctions using blockchain smart contracts**

Authors: Omar, Ilhaam A. Hasan, Haya R. Jayaraman, Raja Salah, Khaled Omar, Mohammed

Publication: Technological Forecasting & Social Change - 2021

They proposed a general framework for decentralized auctions leveraging Ethereum smart

10

contracts to trace and track bids, decentralized storage systems to upload documents related to bidding and trusted timer oracles that act as gateway between smart contract and external data feeds. In the proposed solution, they also developed detailed algorithms that define the working principles of the Smart contract for the auction process. We present detailed cost analysis of the solution to demonstrate economic feasibility, providing a secure, transparent and reliable approach to online auctions.

**Paper 19: Decentralized Cloud Storage Using Blockchain**

Authors: Shah, Meet, Mohammedhasan, Shaikh, Vishwajeet, Mishra, and Grinal, Tuscano.

Publication: IEEE – 2020

In this application the user’s file is encrypted and stored across multiple peers in the network using the Inter Planetary File System protocol. Inter Planetary File System protocol creates hash value. The hash value indicates the path of the file and is stored in the blockchain. They also focus on decentralized secure data storage, high availability of data, and efficient utilization of storage resources. Implemented system uses the AES 256bit encryption algorithm to encrypt the data ensuring the confidentiality of the user’s data. Encrypted data is then distributed and stored across peers in the network using the Inter Planetary File System protocol. 2 preconditions for the smart contract to execute were have Enough Space is available in the network to store files and The user has sufficient wallet balance to pay the peers.

**Paper 20: A Blockchain and Self-Sovereign Identity Empowered Digital Identity Platform**

Authors: Bandara, Eranga, Xueping, Liang, Peter, Foytik, Sachin, Shetty, and Kasun De, Zoysa

Publication: IEEE – 2021

This app Sora is a mobile app-based identity system that utilizes blockchain technology to create a secure protocol for storing encrypted personal information and sharing verifiable claims about personal information. In this system hash values of users’ personal information encrypted with a cryptographic key and published in the blockchain. The system built on top of Hyperledger Iroha permissioned blockchain. This Sora mobile apps allow the user to generate their cryptographic key, input their data, encrypt it, and publish salted hashes of their data to the blockchain. Users can then share their personal information of their own volition to institutions, such as banks or other organizations.

11

Chapter 3

**Implementation**

In this chapter we will discuss the implementation part of the project. This will cover the specification used to develop the project, design approach, module description, UML diagrams which are necessary to understand the project and testing of the project. At the end it will also cover the implemented code which can be used for further development.

**3.1 TECHNICAL SPECIFICATION**

**Hardware Specifications**

* Processor: Core i3-10300H
* Ram: 4GB
* Hard Disk Space: 8GB

**Software Requirements**

* Operating System: Windows 8
* Frameworks: ReactJS, ViteJS, NodeJS, Bootstrap, Web3, HardHat.

**3.2 DESIGN APPROACH AND DETAILS**

3.2.1 System Architecture

The Decentralized application is made up of couple of modules such as front end, back end, smart contracts, block chain deployment. The modules together make up the peer to peer connected decentralised chat application. The contract owner deployed the application on the block chain by the use of writing a smart contract. The other modules namely front end and back end of the application are stored in the Database of the contract owner. The transactions being done (in this case operations like sending messages, receiving messages, clearing inbox) are not stored in any database and are written in the blockchain by the creation of new block. Metamask is being used for managing the peer’s wallet and ether economics. All the transaction will be hashed by SHA3-256 Algorithm to make every transaction secure while attaining Decentralization.

12

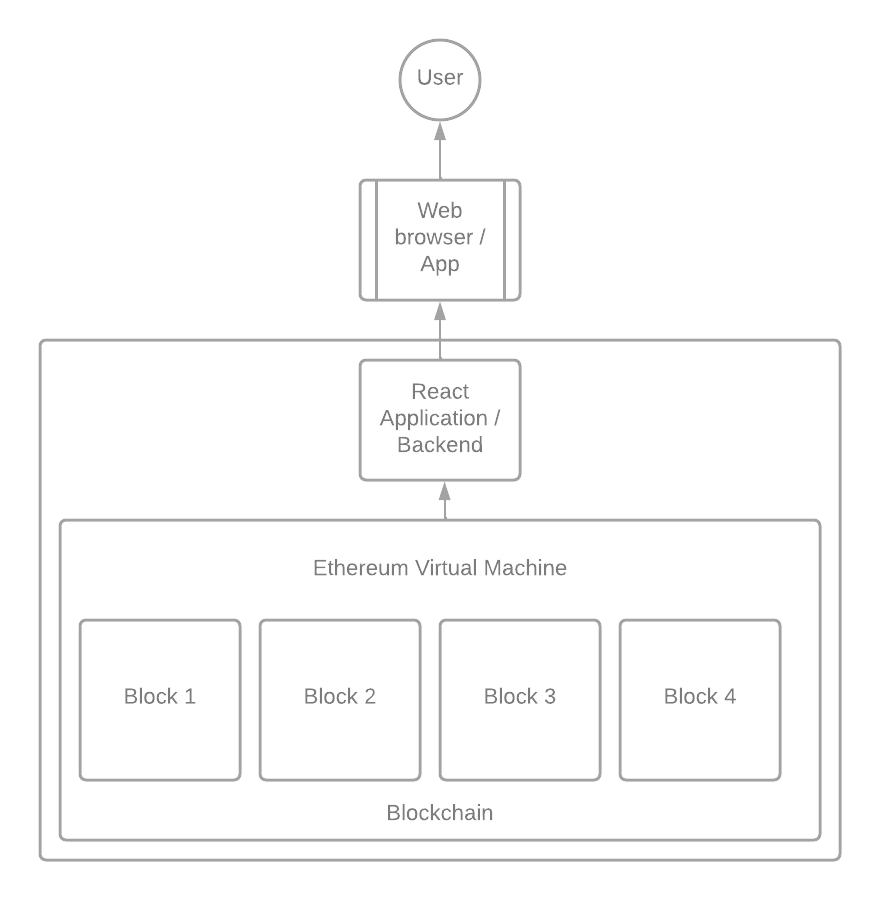
****

Figure 3.1

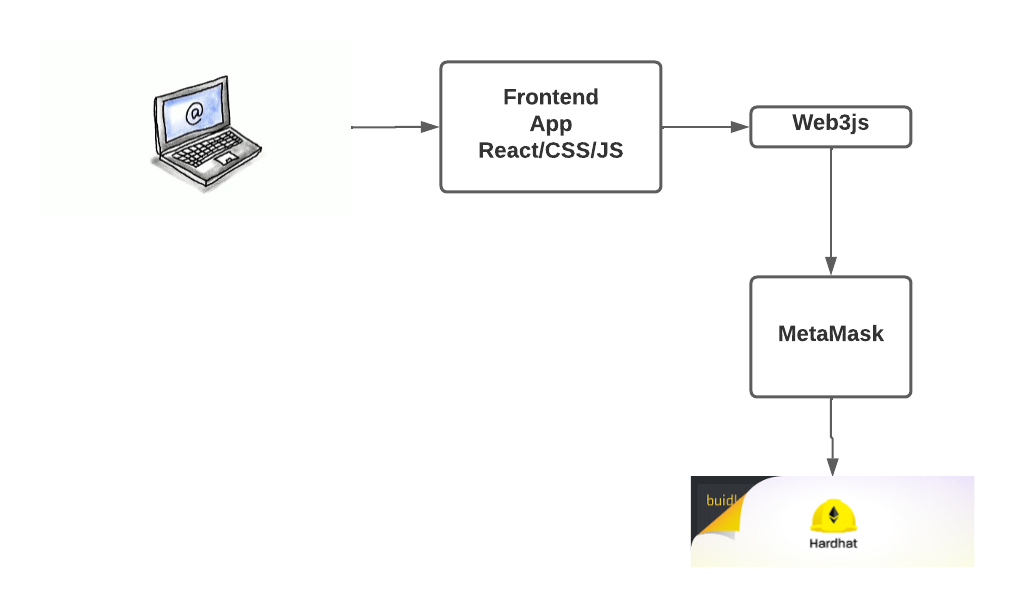
****

Figure 3.2

13

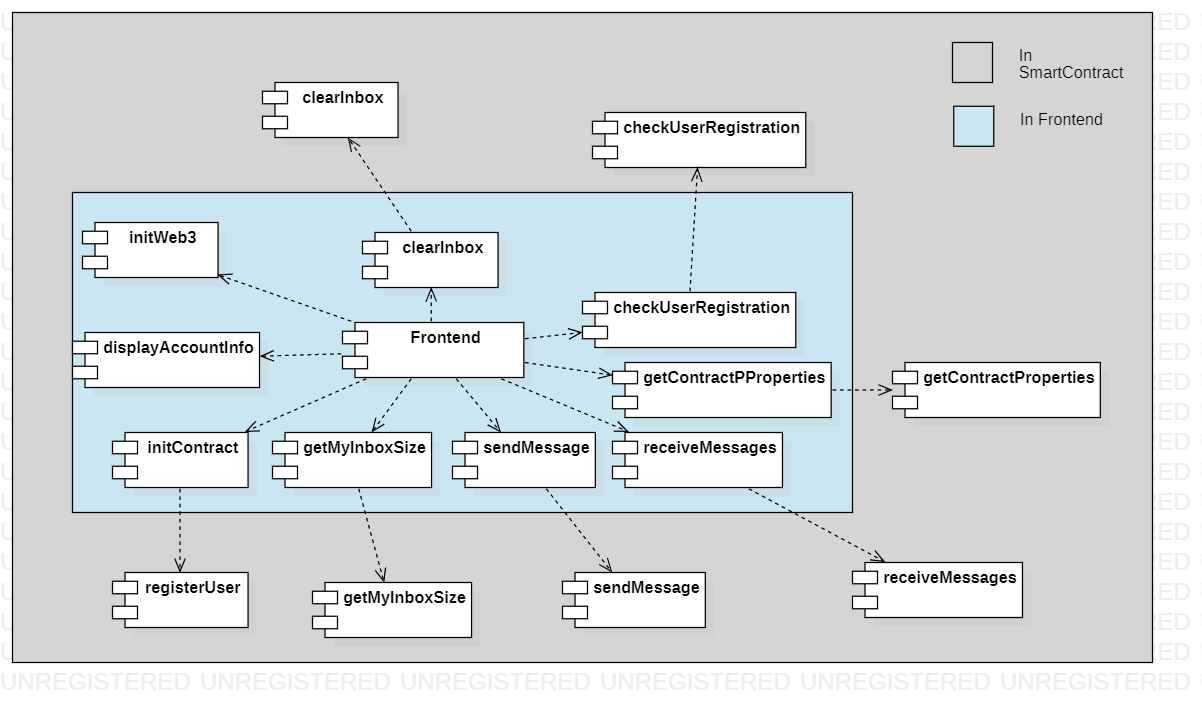
****

Figure 3.3

**3.3 MODULE DESCRIPTION**

The Decentralised application is built on numerous frameworks and development environment. The application uses frameworks like web3 and more to achieve the decentralization to provide a safe and secure messaging platform. Now the modules which make up the application are discussed in detail.

Front End:

The web page which the user interacts to make use of this Decentralised application is mainly built upon using ReactJS, ViteJS, HTML5, Tailwind and JavaScript. The page which the user will be interacting is named as index.html. Index.html uses CSS from the file app.css. In particular to CSS the front end makes use of the CSS framework called Tailwind CSS which has pre-written functions to build a application. The user also interacts with Metamask which is the crypto currency wallet for the Ethereum network. It handles all the transaction and manages the fee charged for each transaction done by the user.

Components involved in the Frontend Vite JS.

**Base Component** - App

This will handle all the other components by providing all the necessary information like contract Address and Abi to the sub components. This also binds all the other components together. This also renders the main components App Dashboard.

14

**Create a basic structure of the web page** – Form Dashboard

This component helps in making the AppBar and the application name and other details which will be displayed in the website. Later this renders the Display details component.

**Display Important details** - DisplayDetails.js

This component deals in rendering all the necessary information like Contract address, Contract owner, Ethereum address and the current balance in the Account. Next it renders the components like ListUserDirectory, SendForm and ListItemText.

**Display all the available users** – List User Directory

This component will render a dropdown select component where all the available/ registered users address will be displayed. So this will allow the current user to copy the address of the receivers address so that it can be used to send the message.

**Get necessary inputs for sending message** – Send Form

This is a very a very simple component which help in getting the receivers address from the user. This will render a Text field where user can input the recipient address.

**Gets the contents of the message and handles sending message** - SendMessageForm.js

In this component it will render a text field to input the text message and a button to send the message to the recipient.

**Display the received messages** – Render Received Messages

In this component it renders all the received message from the other users. It will also contain a button for clearing the inbox.

At the end it also displays the status of the connection between metamask and the browser.

Upon refresh of page the following happing in order:

1. **Initialise Web3** Provider;

Initialize web3 and set the provider to the testRPC. Sets the provider we want from Web3.providers. Get the initial account balance so it can be displayed.

1. **Load Contract**;

Get the necessary contract artifact file and use it to instantiate a truffle contract abstraction. Set the provider for our contract.

15

1. **Load Account Information**;

This Component helps in displaying their Account Information by retrieving information of the account from the browsers metamask object.

1. **Initialise the account**;

This function check if the user is already registered if not this will register them. So that the address will be added to the list of registered users which could be useful in connecting with other accounts.

1. **get the accounts Inbox Size**;

This will be useful when we display the received messages are displayed.

1. **Sending Message**

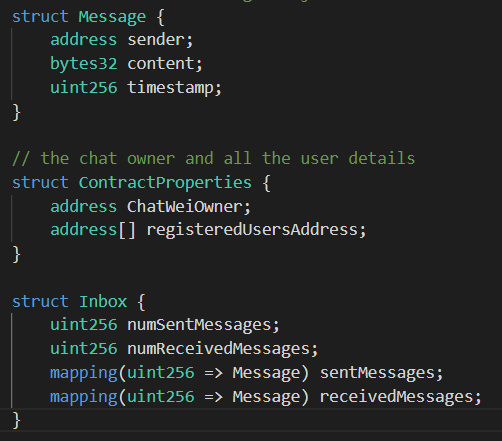
This module will handle on sending the message by converting the string message to byte32 format.

1. **Receive Messages**;

This module will be used for reviving the received messages for the user when refresh button is clicked.

Smart Contracts:

Hardhat is a development environment to compile, deploy, test, and debug your Ethereum software. It helps developers manage and automate the recurring tasks that are inherent to the process of building smart contracts and decentralised Apps, as well as easily introducing more functionality around this workflow. This means compiling, running and testing smart contracts at the very core. The Contracts are written in a programming language called solidity which used for implementing smart contract on various block chain platforms.



Structure objects figure 3.4

16

ChatWei Contract Is the name of the contract Inside in we will have 3 struct each for Message, Contract Properties and Inbox.

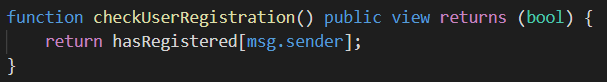
Let’s see in detail for those components the Message Structure Consist of sender Address, Content in bytes32 and timestamp. In Contract Properties Structure it contains Chat Owner address and list of all registered users. At last, in Inbox, it contains number of sent and received messages, a dictionary of sent message and received messages. Then a map of Inbox is created for each user.

In the constructor of the chatwei first we will be trying to register a new user if they don’t exist and assign the owner of the smart contract in the contract properties.

Let’s go deep in to all the functions:

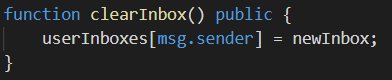
**Check User is already Registered**:

It will check if the user already exists and returns a bool value. In case of true it will represent the user is registered.



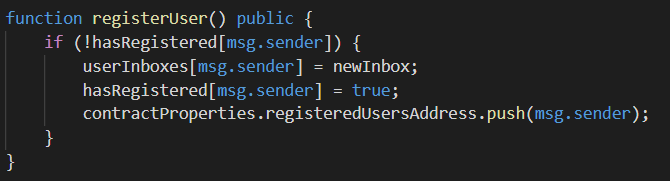
**Clear all the data of a user’s Inbox**:

This function will clear the inbox of the user.



**Registering new User:**

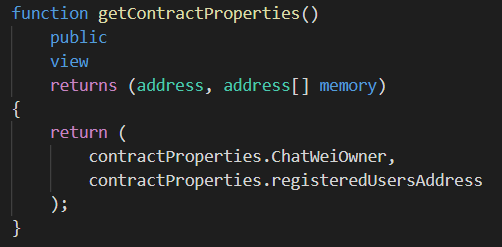
Checks if the user exists, if not exist it will assign a new inbox and add the user address to the list of registered users.



17

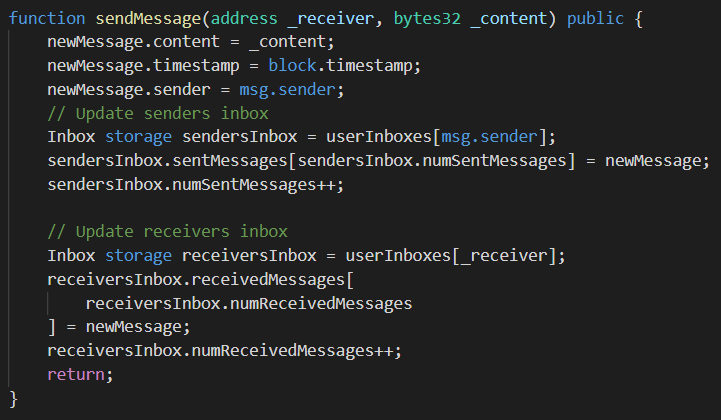
**Retrieve the Contract Properties:**

this will return the owner of the contract and the list of registered users.



**Send Message:**

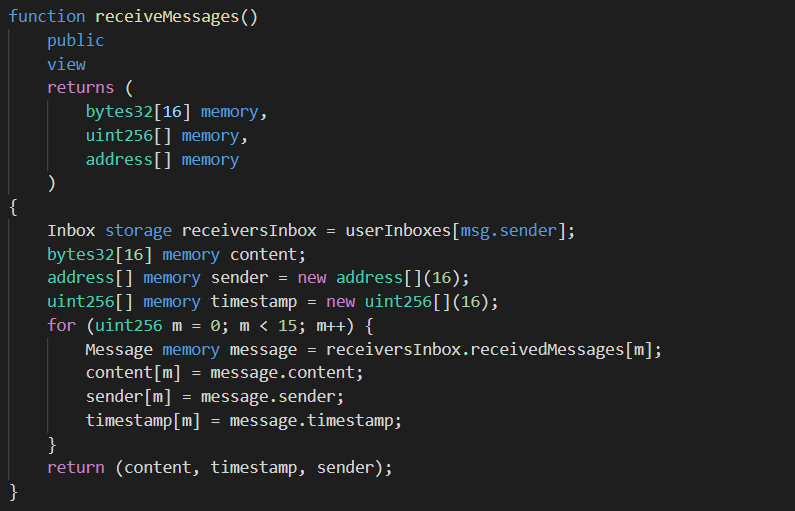
this requires parameters like receiver address and the content of the message. Assign all the required properties like content, timestamp and sender address to the newMessage variable with the properties of the Message Structure. Get the Inbox assigned for the sender and receiver, append the new message to the sentMessages and receivedMessages property respectively.



18

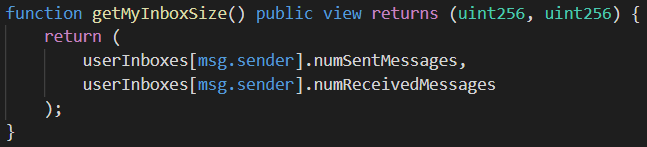
**Receive Messages**:

when the user requests for the messages it received this function will retrieve all the received messages and returns it.



**Get the Size of an Inbox**:

It returns the properties of the user’s Inbox.



19

**3.4 TESTING**

I am going to use remix for the testing of my smart contract.

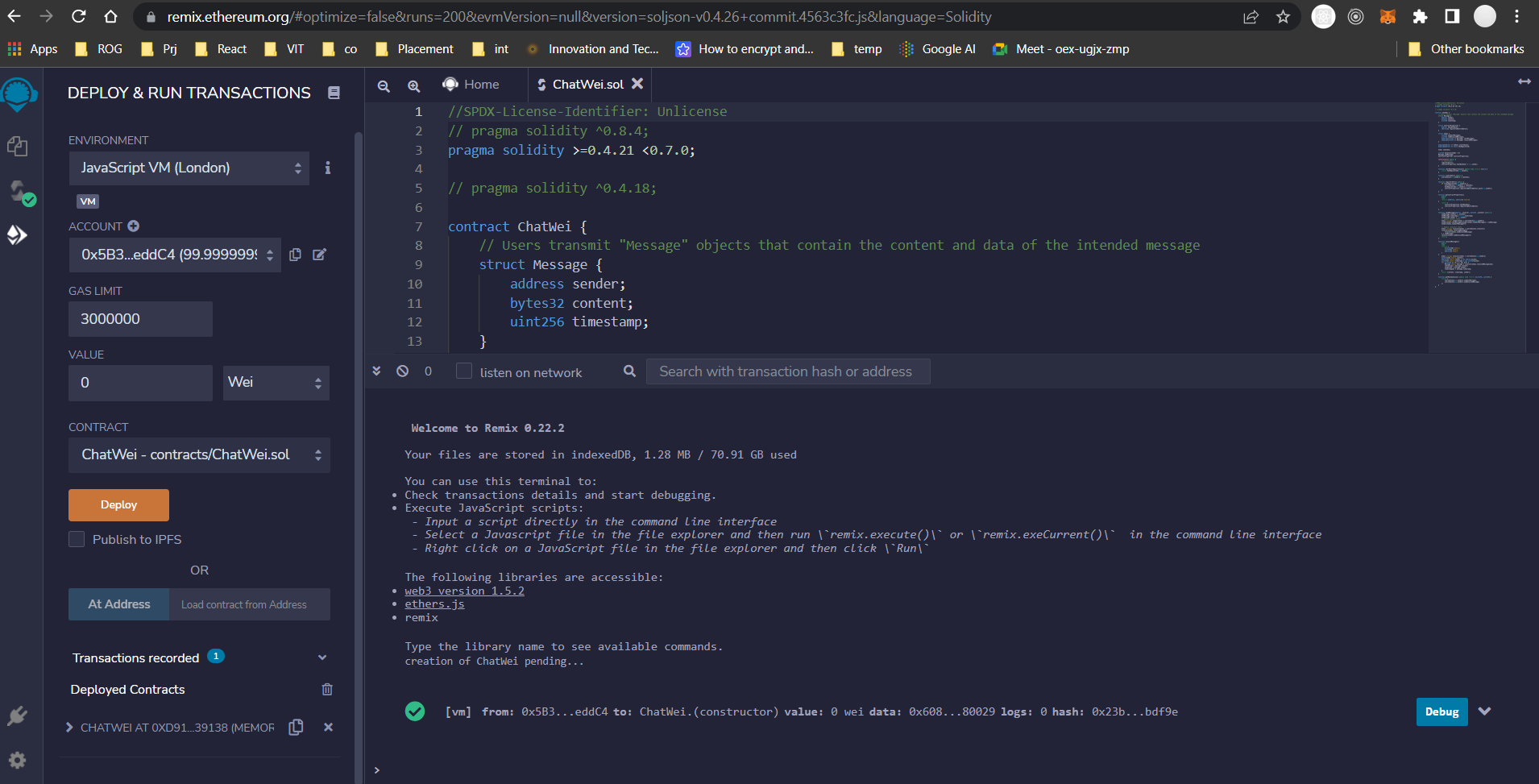


Figure 4

After deploying remix gives me 10 accounts for the testing:

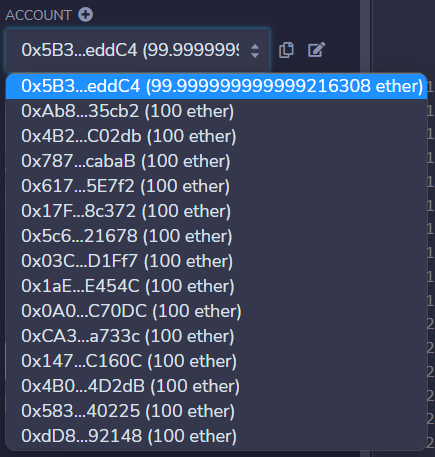


Figure 5

20

All the available functions in the smart contract a listed by the remix for the testing purpose:

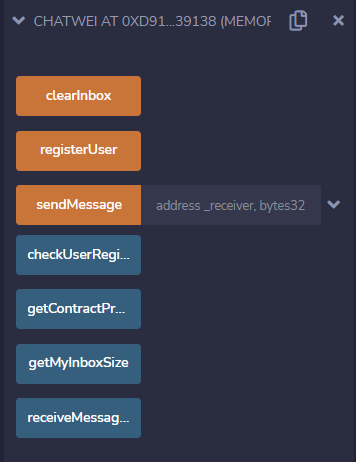


Figure 6

Step 1: So first we should check about the creation of the account. This can be achieved by calling the function checkUserRegistration().

Step 2: lets switch to another account and check the account status. if it is not created then lets add the account by calling the function registerUser().

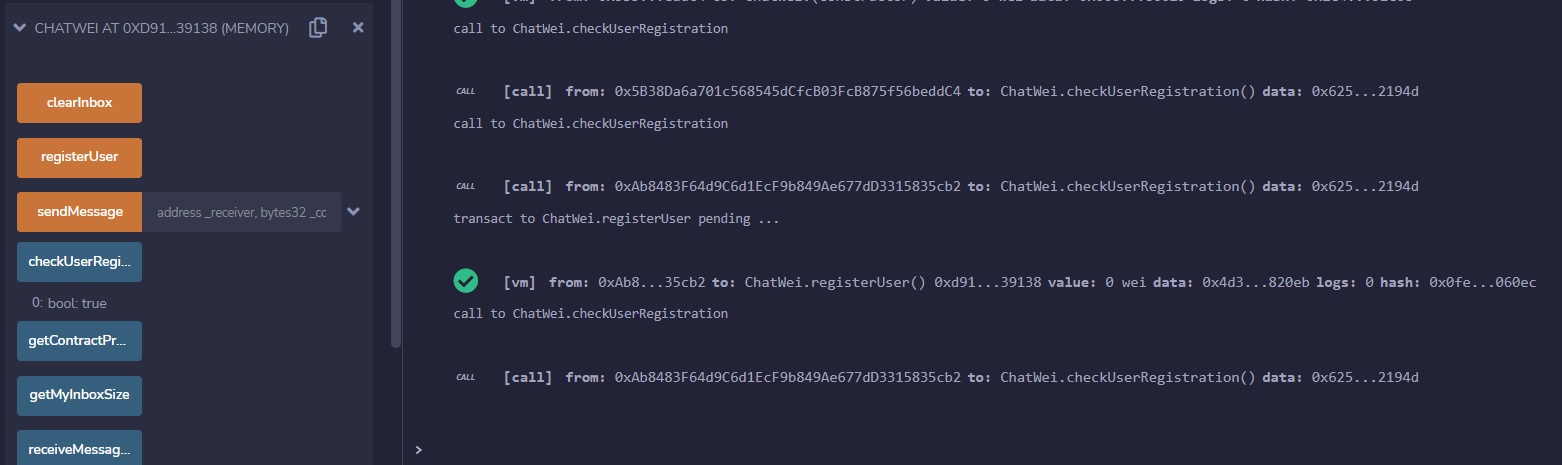


Figure 7

Step 3: Now lets try to send a message to the first account So in order to do that we should get the address of the first account this can be done by calling the function getContractProperties().

21

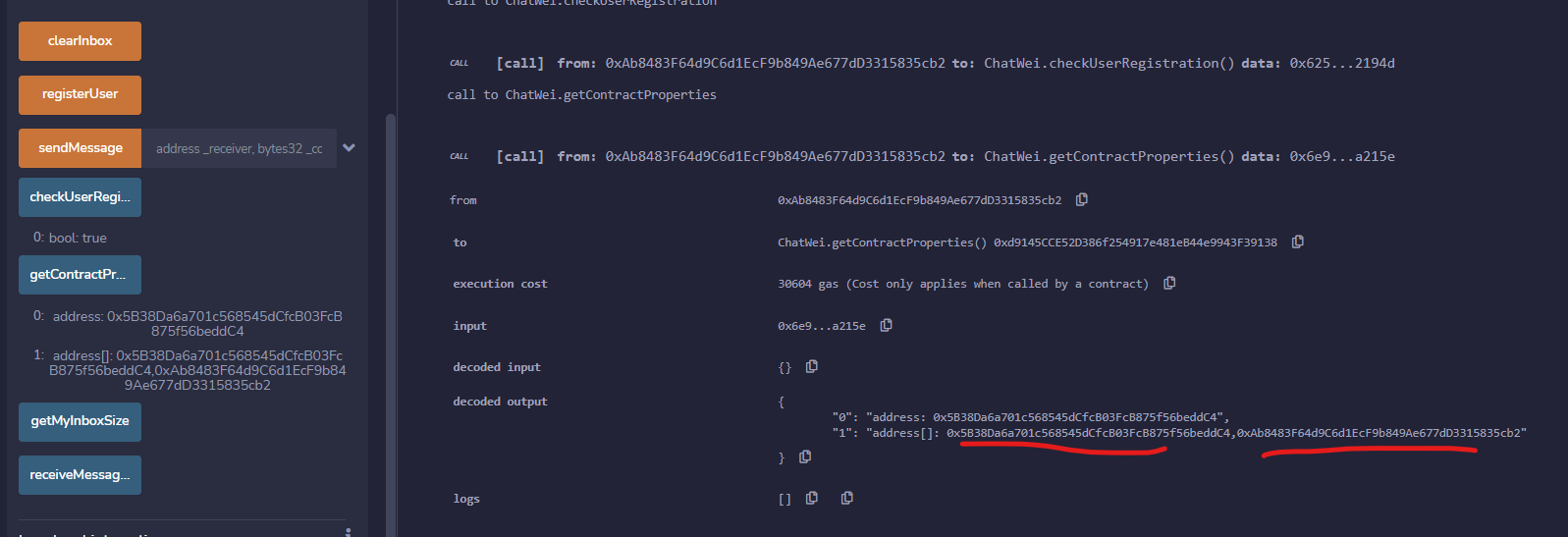


Figure 8

Step 4: here we can find the list of all available address in the form of array. So we get the address of the first account .

Step 5: now we need to send the message by filling the \_receiver and \_content with the address of the receiver and the content of the message as byte32 format. And click on transact.

I am using the string “Silver” in byte32 it is “0x53696c7665720000000000000000000000000000000000000000000000000000”.

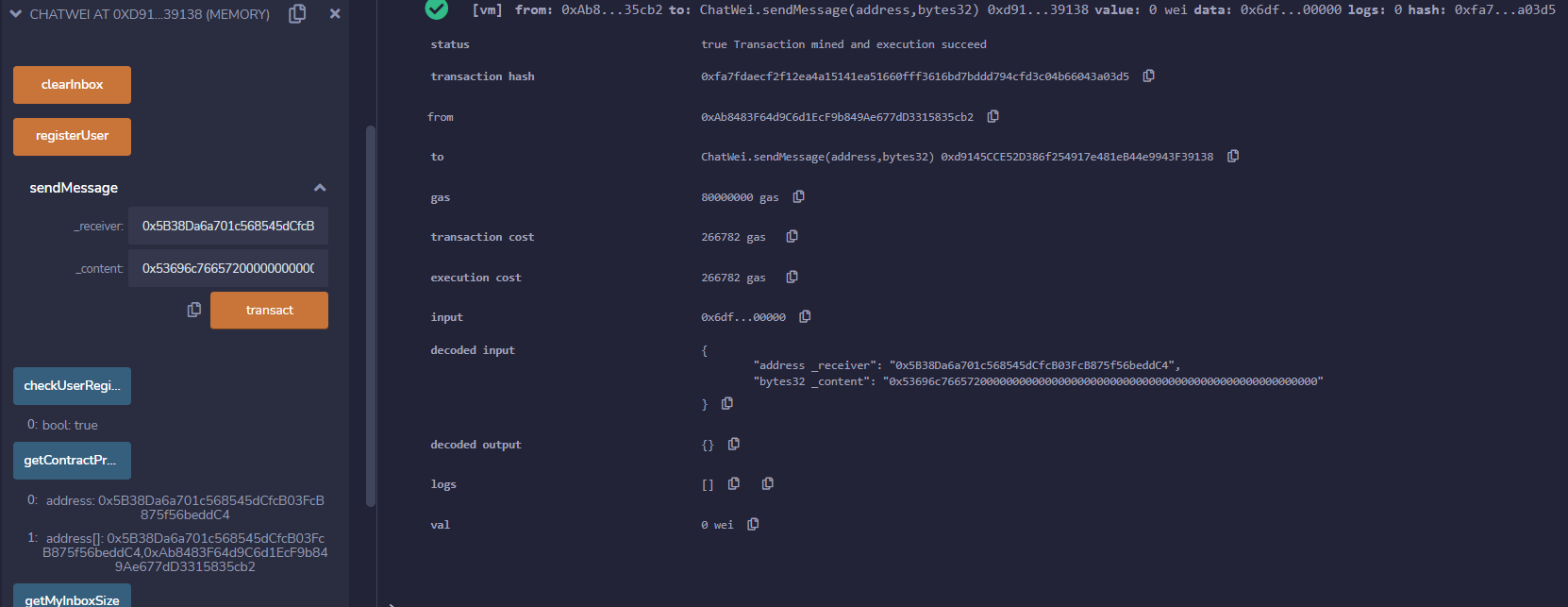


Figure 9

22

Step 6: now let’s get back to the first account.

Step 7: When we click on the receiveMessage() finction it will return the received messages for that account.

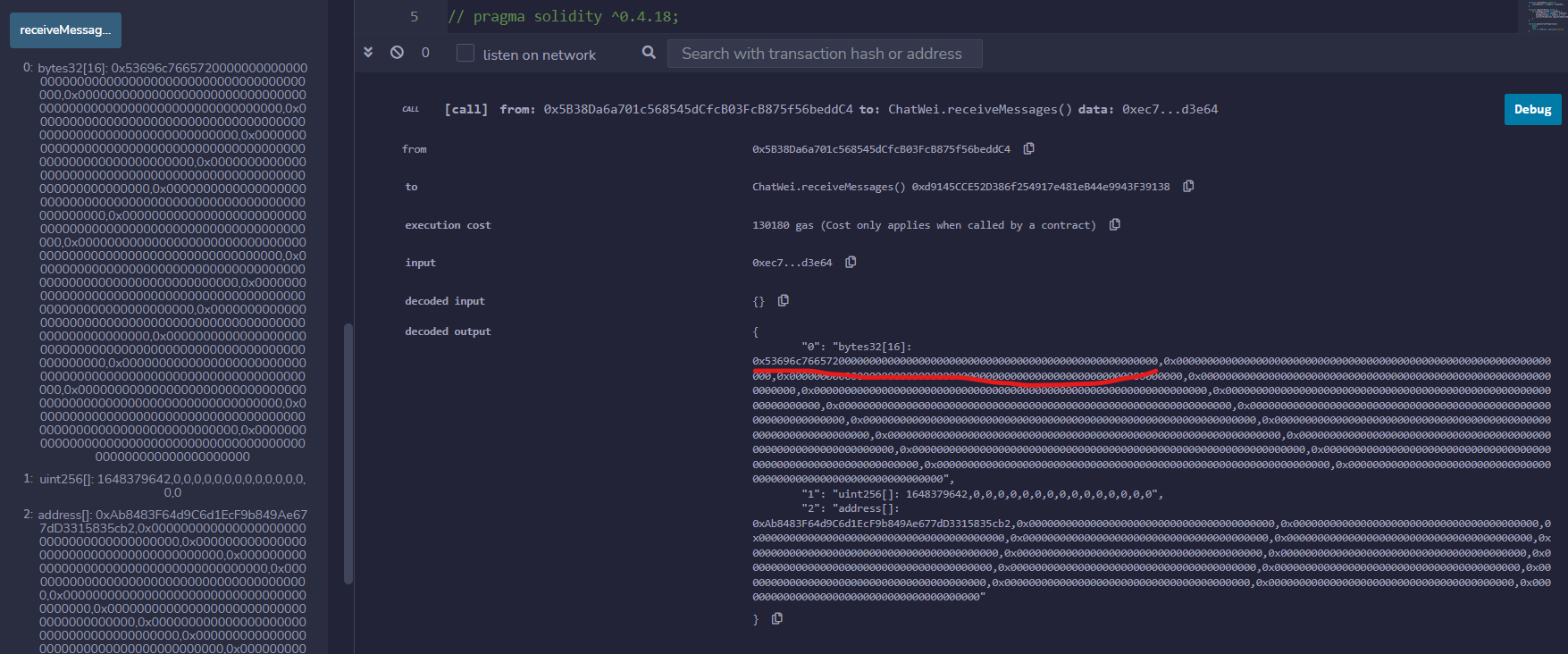


Figure 10

Step 8: now I will again try to do the steps and check the getMyInboxSize() – here I trid to send 3 messages from the account 2 to the first account. Now we can note the count is 3 as expected.

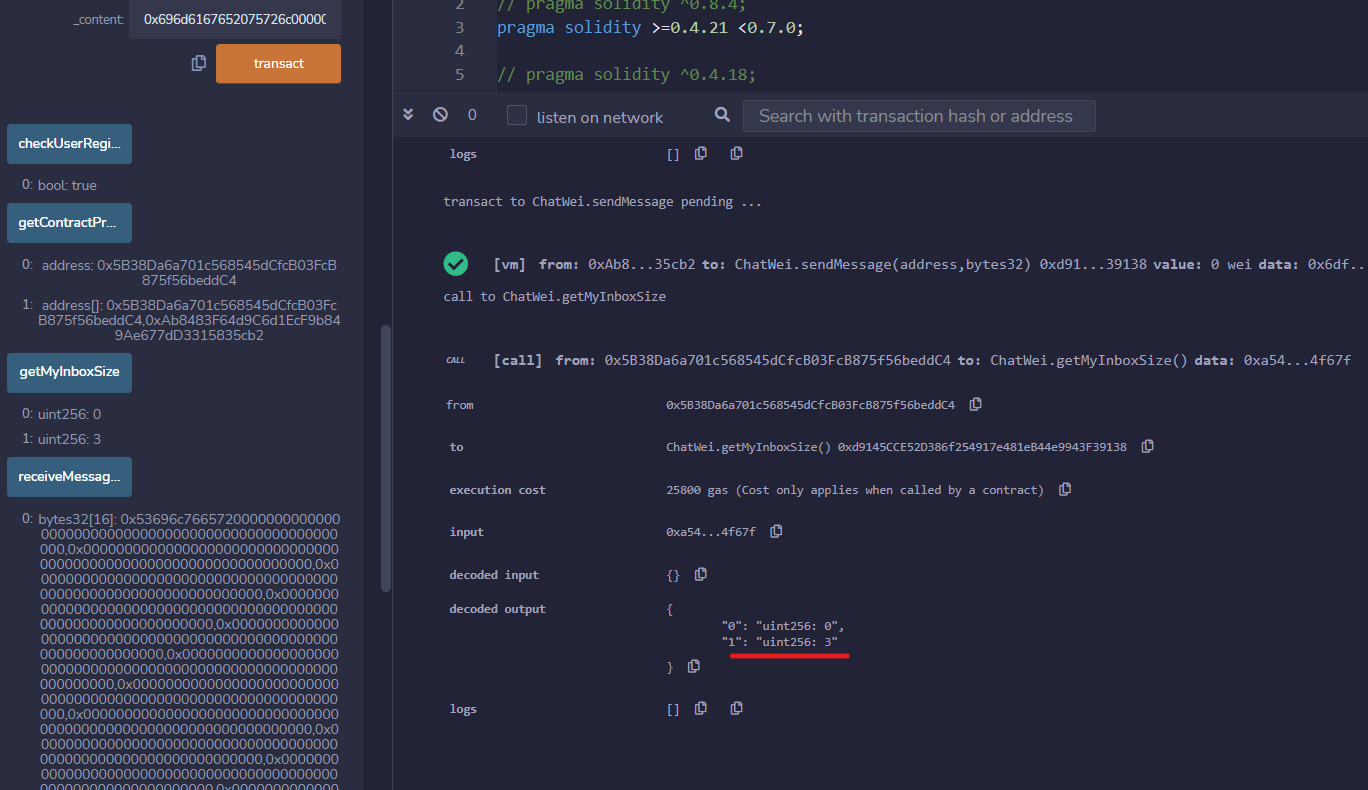


Figure 11

23

Step 9: after trying to clear the inbox by using the function clearInbox() again in checked the inbox size and it was 0 again. So now we can conclude all the functions works well.

Now lets check the frontend:

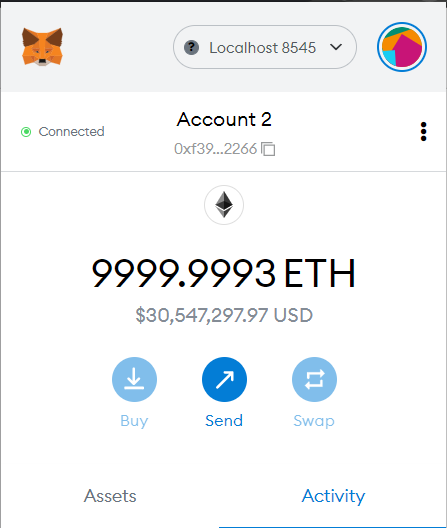


Figure 12

This is the first account by which the contract is deployed.

24

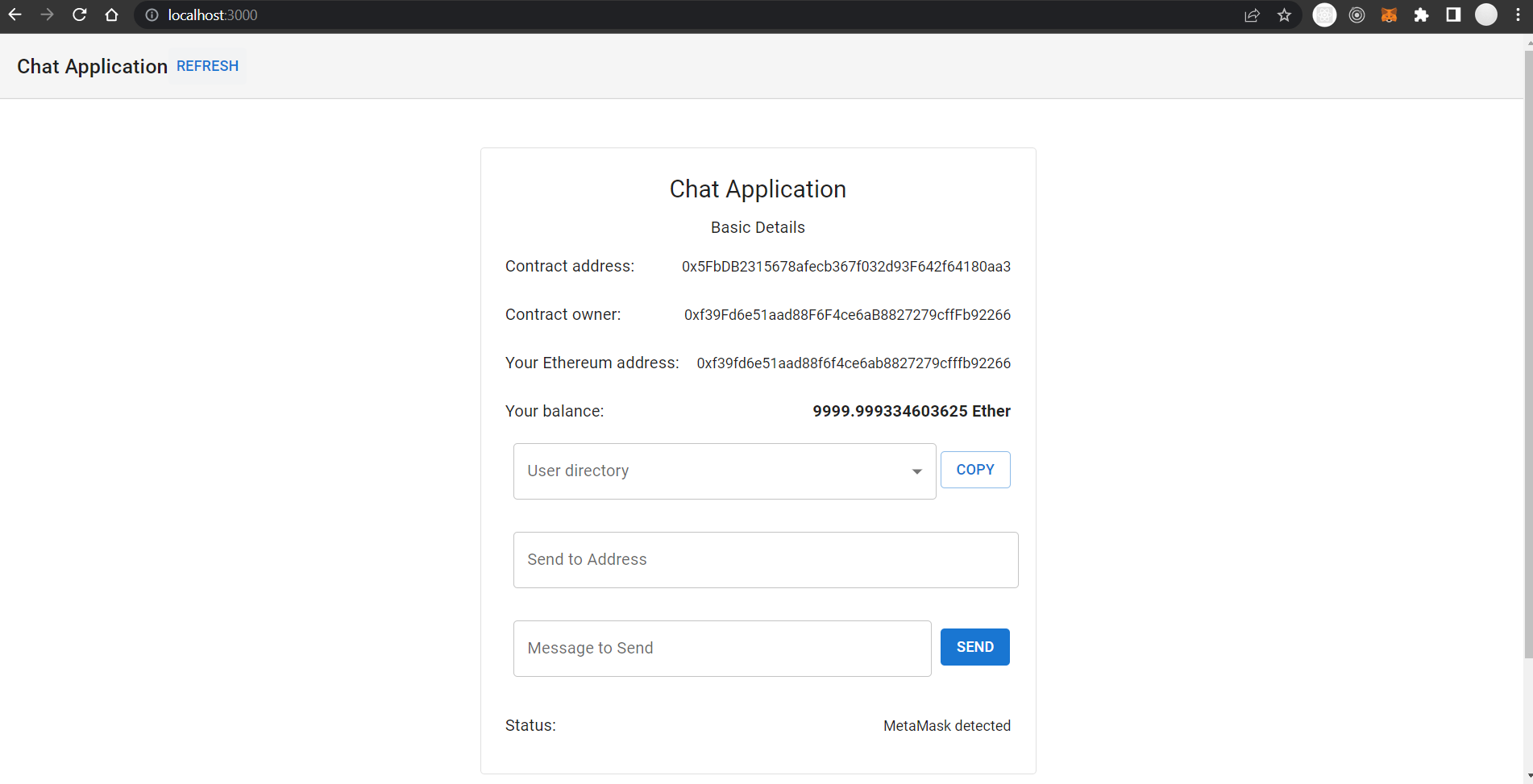


Figure 13

This is an interface of the App.

Step 1: lets login to another account and send messages to the first account.

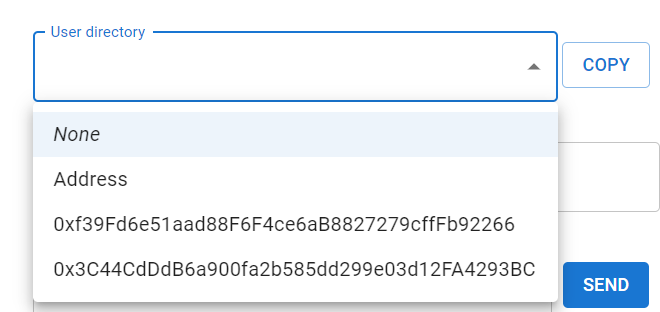


Figure 14

Here we can find all the registered account address.

25

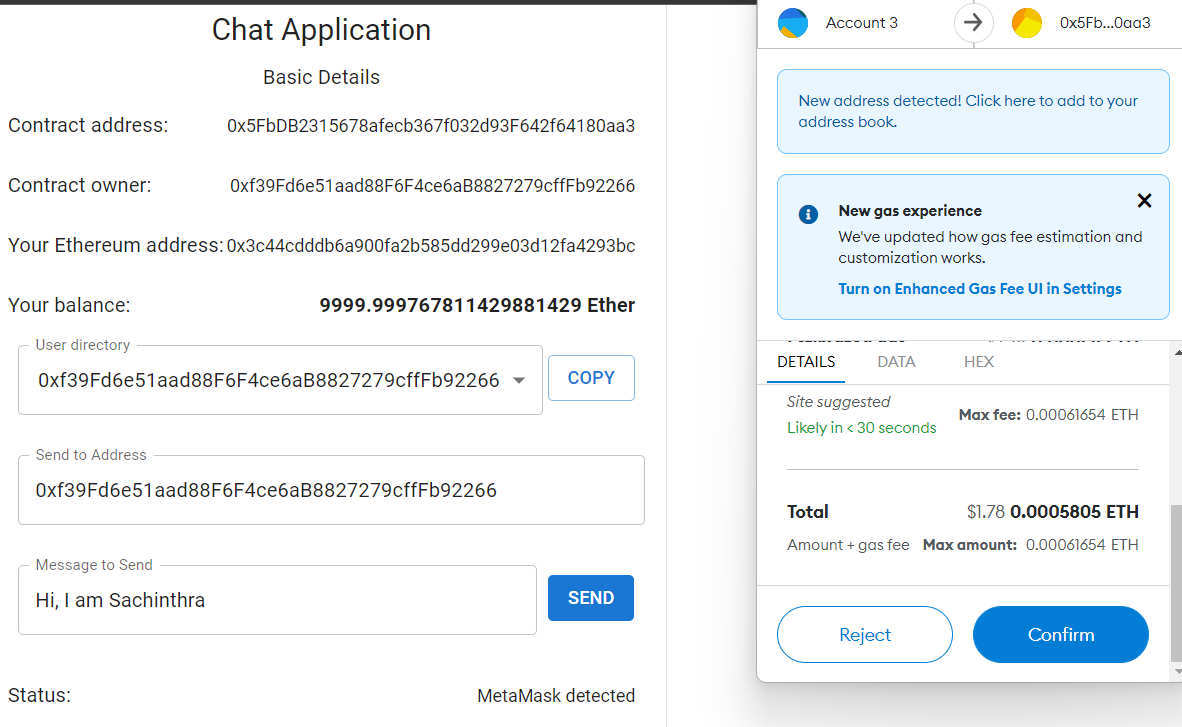


Figure 15 : Sending message.

Step 2: go to the first account and refresh the received message.

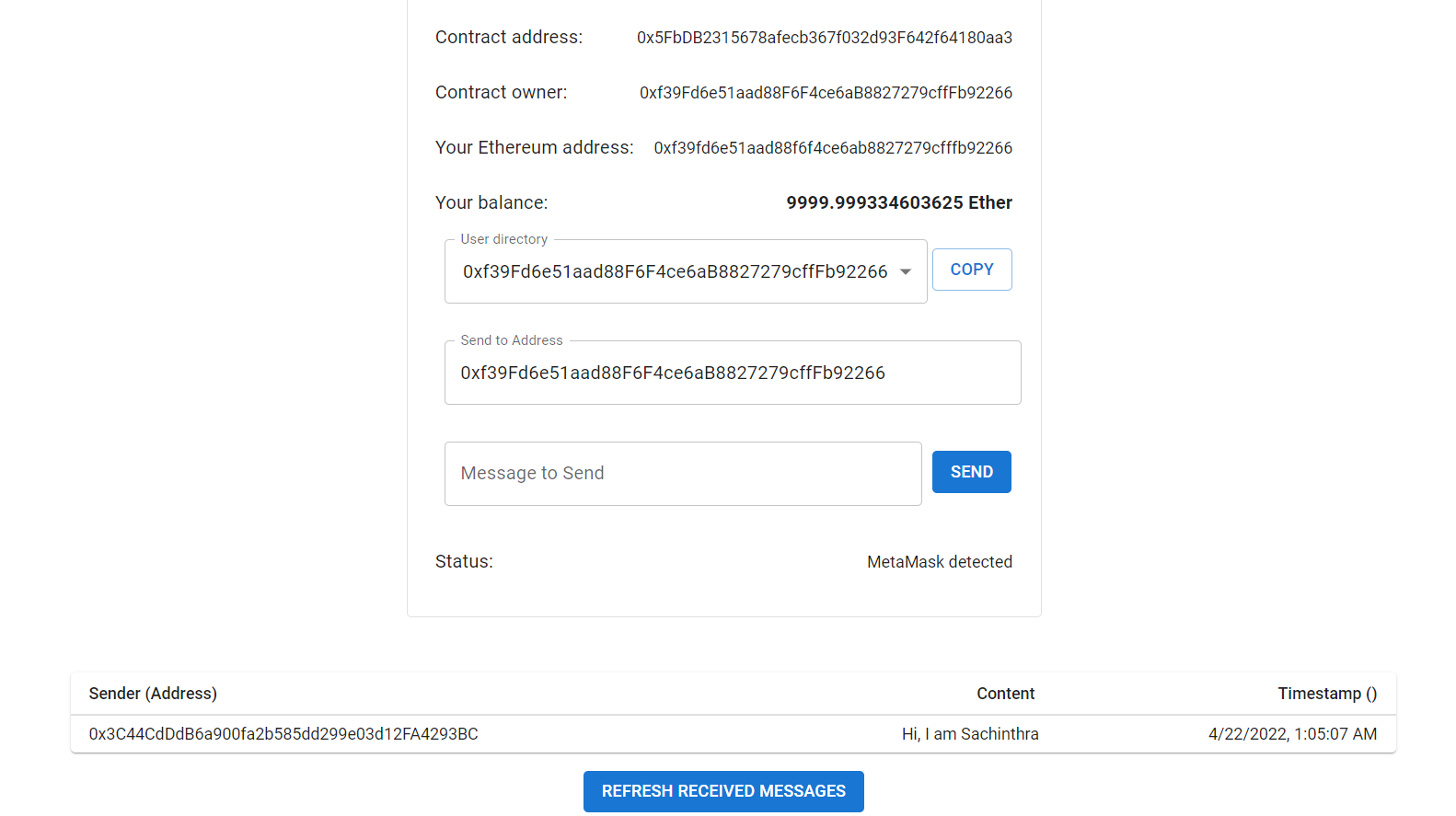


Figure 16 : Now we can see the received message from the other account.

26

**3.5 UML DIAGRAM**

**Use Case Diagram**

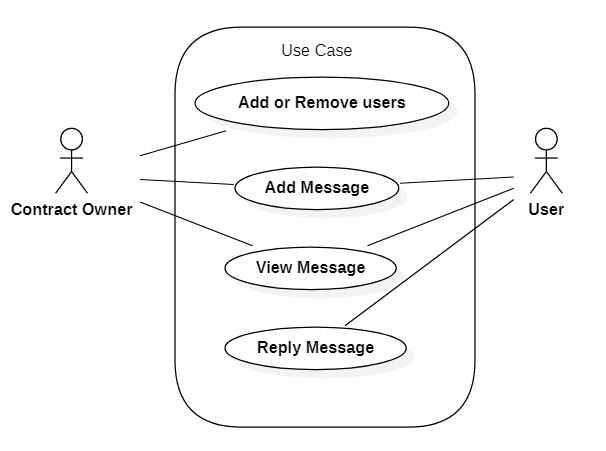
****

Figure 17

**Activity Diagram**

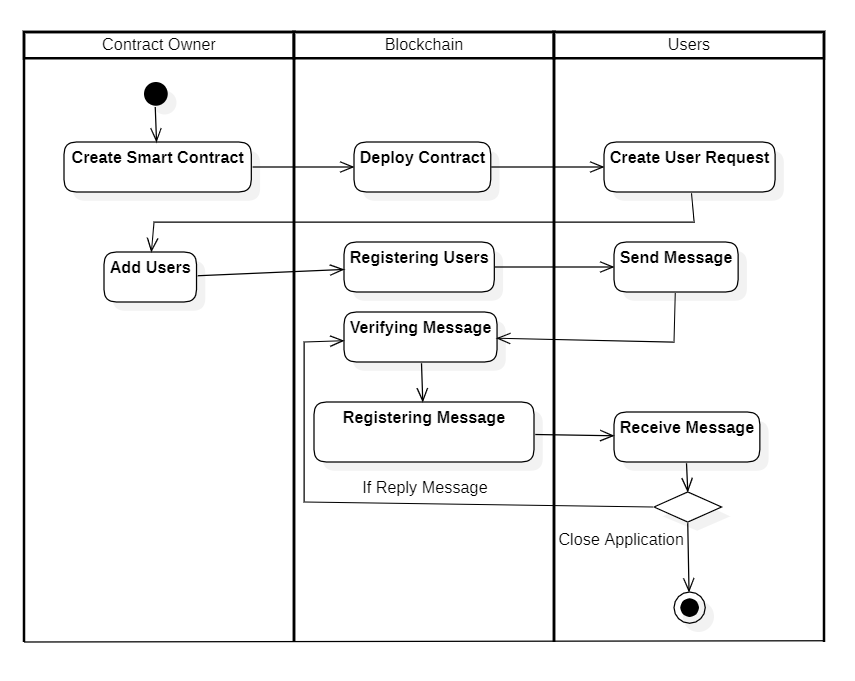
****

Figure 18

**27**

**Class Diagram**

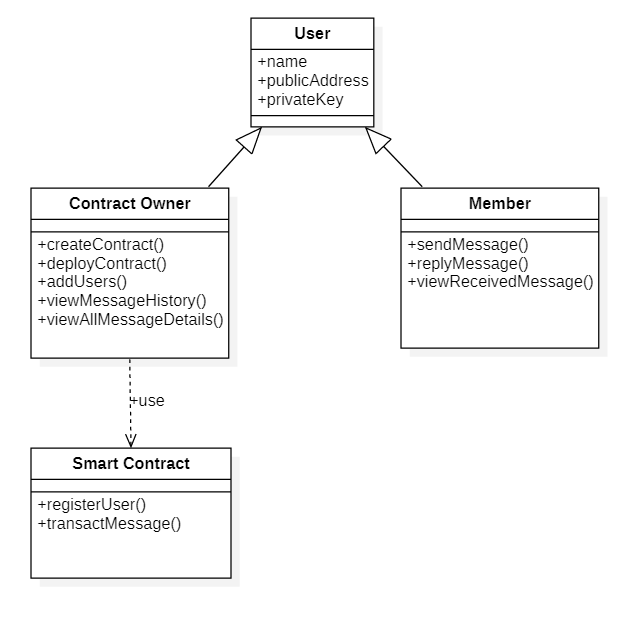
****

Figure 19

28

**3.6 CODE** (GitHub: <https://github.com/sachinthra/blockchain-chat-application.git>)

**Smart Contract Code**

pragma solidity >=0.4.21 <0.7.0;

// pragma solidity ^0.4.18;

contract ChatWei {

    // users transmit "Message" objects that contain the content and data of the message.

    struct Message {

        address sender;

        bytes32 content;

        uint256 timestamp;

    }

    // the chat owner and all the user details

    struct ContractProperties {

        address ChatWeiOwner;

        address[] registeredUsersAddress;

    }

    struct Inbox {

        uint256 numSentMessages;

        uint256 numReceivedMessages;

        mapping(uint256 => Message) sentMessages;

        mapping(uint256 => Message) receivedMessages;

    }

    mapping(address => Inbox) userInboxes;

    mapping(address => bool) hasRegistered;

    Inbox newInbox;

    uint256 donationsInWei = 0;

    Message newMessage;

    ContractProperties contractProperties;

    constructor() public {

        registerUser();

        contractProperties.ChatWeiOwner = msg.sender;

    }

29

    function checkUserRegistration() public view returns (bool) {

        return hasRegistered[msg.sender];

    }

    function clearInbox() public {

        userInboxes[msg.sender] = newInbox;

    }

    function registerUser() public {

        if (!hasRegistered[msg.sender]) {

            userInboxes[msg.sender] = newInbox;

            hasRegistered[msg.sender] = true;

            contractProperties.registeredUsersAddress.push(msg.sender);

        }

    }

    function getContractProperties()

        public

        view

        returns (address, address[] memory)

    {

        return (

            contractProperties.ChatWeiOwner,

            contractProperties.registeredUsersAddress

        );

    }

    function sendMessage(address \_receiver, bytes32 \_content) public {

        newMessage.content = \_content;

        newMessage.timestamp = block.timestamp;

        newMessage.sender = msg.sender;

        // Update senders inbox

        Inbox storage sendersInbox = userInboxes[msg.sender];

        sendersInbox.sentMessages[sendersInbox.numSentMessages] = newMessage;

        sendersInbox.numSentMessages++;

        // Update receivers inbox

        Inbox storage receiversInbox = userInboxes[\_receiver];

        receiversInbox.receivedMessages[

            receiversInbox.numReceivedMessages

        ] = newMessage;

        receiversInbox.numReceivedMessages++;

        return;

    }

30

    function receiveMessages()

        public

        view

        returns (

            bytes32[16] memory,

            uint256[] memory,

            address[] memory

        )

    {

        Inbox storage receiversInbox = userInboxes[msg.sender];

        bytes32[16] memory content;

        address[] memory sender = new address[](16);

        uint256[] memory timestamp = new uint256[](16);

        for (uint256 m = 0; m < 15; m++) {

            Message memory message = receiversInbox.receivedMessages[m];

            content[m] = message.content;

            sender[m] = message.sender;

            timestamp[m] = message.timestamp;

        }

        return (content, timestamp, sender);

    }

    function getMyInboxSize() public view returns (uint256, uint256) {

        return (

            userInboxes[msg.sender].numSentMessages,

            userInboxes[msg.sender].numReceivedMessages

        );

    }

}

Frontend APP component

import { useEffect, useState } from "react";

import { ethers } from "ethers";

import { useSelector, useDispatch } from "react-redux";

import "./App.css";

import FormDashBoard from "./components/FormDashBoard";

import ChatWei from "./contracts/ChatWei.json";

import { getSCprops } from "./reduxProcess/actions/scProps";

31

import { getUserProps } from "./reduxProcess/actions/userPropsAction";

import { setAppHandler } from "./reduxProcess/actions/AppHandlerAction";

const contractAddress = "0x5FbDB2315678afecb367f032d93F642f64180aa3";

function App() {

  const { LoadSCpropReducer, UserpropsReducer } = useSelector((state) => state);

  const dispatch = useDispatch();

  useEffect(() => {

    checkWalletIsConnected();

  }, []);

  const checkWalletIsConnected = async () => {

    const { ethereum } = window;

    if (ethereum) {

      console.log("ethereum: " + ethereum);

    } else {

      console.log("Object not found");

      dispatch(

        setAppHandler({ status: false, message: "MetaMask Not detected" })

      );

      return;

    }

    const accounts = await ethereum.request({ method: "eth\_accounts" });

    if (accounts.length !== 0) {

      dispatch(setAppHandler({ status: true, message: "MetaMask detected" }));

      const account = accounts[0];

      console.log("Found an authorized account: ", account);

      // setCurrentAccount(account);

      // getbalance(account)

      const balance = await window.ethereum.request({

        method: "eth\_getBalance",

        params: [account, "latest"],

      });

32

dispatch(

        getUserProps({

          account: account,

          balance: ethers.utils.formatEther(balance),

        })

      );

      fetchChatWeis();

    } else {

      console.log("No authorized account found");

      dispatch(

        setAppHandler({ status: false, message: "No authorized account found" })

      );

    }

  };

  const fetchChatWeis = async () => {

    const { ethereum } = window;

    if (!ethereum) {

      alert("Please install MetaMask!");

      return;

    }

    const provider = new ethers.providers.Web3Provider(ethereum);

    const signer = provider.getSigner();

    const contract = new ethers.Contract(

      contractAddress,

      ChatWei.abi,

      signer

    );

    const isUSerRegistered = await contract.checkUserRegistration();

    console.log("isUSerRegistered: "+isUSerRegistered);

    if(!isUSerRegistered) {

      // await contract.registerUser();

      const result = await contract.registerUser().then(() => console.log("submitted"))

        .catch(e => console.log(e));

    }

    const chatWei = await contract.getContractProperties();

    dispatch(getSCprops({ contract, chatWei }));

   };

33

  return (

    <div className="App">

      <FormDashBoard checkWalletIsConnected={checkWalletIsConnected} />

    </div>

  );

}

export default App;

Form Dashboard Component – FormDashBoard

import \* as React from "react";

import CssBaseline from "@mui/material/CssBaseline";

import AppBar from "@mui/material/AppBar";

import Container from "@mui/material/Container";

import Toolbar from "@mui/material/Toolbar";

import Paper from "@mui/material/Paper";

import Button from "@mui/material/Button";

import Typography from "@mui/material/Typography";

import { createTheme, ThemeProvider } from "@mui/material/styles";

import DisplayDetails from "./DisplayDetails";

import ReceiveMessages from "./receiveMessages";

const theme = createTheme();

export default function FormDashBoard({ checkWalletIsConnected }) {

  const [activeStep, setActiveStep] = React.useState(0);

  return (

    <ThemeProvider theme={theme}>

      <CssBaseline />

      <AppBar

        position="absolute"

        color="default"

        elevation={0}

        sx={{

          position: "relative",

          borderBottom: (t) => `1px solid ${t.palette.divider}`,

        }}

      >

34

<Toolbar>

          <Typography variant="h6" color="inherit" noWrap>

            Chat Application

          </Typography>

          <Button onClick={() => checkWalletIsConnected()} variant="text">

            Refresh

          </Button>

        </Toolbar>

      </AppBar>

      <Container component="main" maxWidth="sm" sx={{ mb: 4 }}>

        <Paper

          variant="outlined"

          sx={{ my: { xs: 3, md: 6 }, p: { xs: 2, md: 3 } }}

        >

          <React.Fragment>

            <React.Fragment>

              <Typography variant="h5" gutterBottom>

                Chat Application

              </Typography>

              <Typography variant="subtitle1">Basic Details</Typography>

            </React.Fragment>

            <React.Fragment>

              <DisplayDetails />

            </React.Fragment>

          </React.Fragment>

        </Paper>

      </Container>

      <Container>

        <ReceiveMessages />

      </Container>

    </ThemeProvider>

  );

}

Display Contract details – DisplayDetails.js

import \* as React from "react";

import Typography from "@mui/material/Typography";

import List from "@mui/material/List";

import ListItem from "@mui/material/ListItem";

import ListItemText from "@mui/material/ListItemText";

import Grid from "@mui/material/Grid";

import ListUserDirectory from "./ListUserDirectory";

35

import SendForm from "./SendForm";

import Button from "@mui/material/Button";

import SendMessageForm from "./SendMessageForm";

import { useSelector,useDispatch } from "react-redux";

import {setSendInfo} from "../reduxProcess/actions/SendInfoAction"

export default function DisplayDetails() {

  const [toUser, setToUser] = React.useState("");

  const [toUserAddress, setToUserAddress] = React.useState("");

  const { LoadSCpropReducer, UserpropsReducer, AppHandlerReducer } =

    useSelector((state) => state);

  const dispatch = useDispatch();

  // console.log(SendInfoReducer);

  return (

    <React.Fragment>

      <List disablePadding>

        <ListItem key={"Contract address:"} sx={{ py: 1, px: 0 }}>

          <ListItemText primary={"Contract address:"} secondary={""} />

          {/\* <Typography variant="body2">{"Could not load"}</Typography> \*/}

          <Typography variant="body2">

            {LoadSCpropReducer.contractAddress || "Could not load"}

          </Typography>

        </ListItem>

        <ListItem key={"Contract owner:"} sx={{ py: 1, px: 0 }}>

          <ListItemText primary={"Contract owner:"} secondary={""} />

          <Typography variant="body2">

            {LoadSCpropReducer.contractOwner || "Could not load"}

          </Typography>

        </ListItem>

        <ListItem key={"Your Ethereum address:"} sx={{ py: 1, px: 0 }}>

          <ListItemText primary={"Your Ethereum address:"} secondary={""} />

          <Typography variant="body2">

            {UserpropsReducer.ethereumAddress || "Could not load"}

          </Typography>

        </ListItem>

        <ListItem sx={{ py: 1, px: 0 }}>

          <ListItemText primary="Your balance:" />

          <Typography variant="subtitle1" sx={{ fontWeight: 700 }}>

            {UserpropsReducer.balance || "0"} Ether

          </Typography>

        </ListItem>

36

</List>

      <Grid container spacing={2}>

        <Grid item xs={12} sm={10}>

          <ListUserDirectory toUser={toUser} setToUser={setToUser} />

        </Grid>

        <Grid item xs={12} sm={2}>

          <Typography variant="h6" gutterBottom sx={{ mt: 2 }}>

            <Button

              variant="outlined"

              disableElevation

              onClick={() => {

                navigator.clipboard.writeText(toUser);

                dispatch(setSendInfo({toUserAddress:toUser}));

              }}

            >

              Copy

            </Button>

          </Typography>

        </Grid>

        <Grid item xs={12} sm={12}>

          <SendForm

          />

        </Grid>

        <SendMessageForm />

        <Grid item xs={12} sm={12}>

          <ListItem key={"Status:"} sx={{ py: 1, px: 0 }}>

            <ListItemText primary={"Status:"} secondary={""} />

            <Typography variant="body2">

              {AppHandlerReducer.statusMessage || "MetaMask Not detected"}

            </Typography>

          </ListItem>

        </Grid>

      </Grid>

    </React.Fragment>

  );

}

37

Displaying User Address Directory - ListUserDirectory.js

import \* as React from 'react';

import InputLabel from '@mui/material/InputLabel';

import MenuItem from '@mui/material/MenuItem';

import FormHelperText from '@mui/material/FormHelperText';

import FormControl from '@mui/material/FormControl';

import Select from '@mui/material/Select';

import { useSelector } from "react-redux";

export default function ListUserDirectory({toUser, setToUser}) {

  const handleChange = (event) => {

    setToUser(event.target.value);

  };

  const { LoadSCpropReducer } = useSelector(

    (state) => state

  );

console.log(toUser)

  return (

    <div>

      <FormControl sx={{ m: 1, minWidth:420 }} >

        <InputLabel id="demo-simple-select-helper-label">User directory</InputLabel>

        <Select

          labelId="demo-simple-select-helper-label"

          id="demo-simple-select-helper"

          value={toUser}

          label="User directory"

          onChange={handleChange}

        >

          <MenuItem value="">

            <em>None</em>

          </MenuItem>

          <MenuItem value={10}>Address</MenuItem>

          {LoadSCpropReducer.usersList.map((user, index) => <MenuItem key={index} value={user}>{user}</MenuItem>)}

        </Select>

        {/\* <FormHelperText>With label + helper text</FormHelperText> \*/}

      </FormControl>

    </div>

  );}

38

Collecting recipient address – SendForm.js

import \* as React from 'react';

import Box from '@mui/material/Box';

import TextField from '@mui/material/TextField';

import { useSelector, useDispatch } from "react-redux";

import {setSendInfo} from "../reduxProcess/actions/SendInfoAction"

export default function SendForm() {

  const { SendInfoReducer } = useSelector((state) => state);

  const dispatch = useDispatch();

  const handleChange = (event) => {

    dispatch(setSendInfo({toUserAddress: event.target.value}));

  };

  return (

    <Box

      component="form"

      sx={{

        '& > :not(style)': { m: 1, width: '100%' },

      }}

      noValidate

      autoComplete="off"

    >

       <TextField id="outlined-basic" label="Send to Address" variant="outlined" value={SendInfoReducer.toUserAddress} onChange={handleChange} />

    </Box>

  );

}

Get message from sender and handles sending – SendMessageForm.js

import \* as React from "react";

import Grid from "@mui/material/Grid";

import Button from "@mui/material/Button";

import Box from "@mui/material/Box";

import Typography from "@mui/material/Typography";

import TextField from "@mui/material/TextField";

import { useSelector, useDispatch } from "react-redux";

import { setSendInfo } from "../reduxProcess/actions/SendInfoAction";

import { ethers } from "ethers";

39

export default function SendMessageForm() {

  const { SendInfoReducer, LoadSCpropReducer } = useSelector((state) => state);

  const dispatch = useDispatch();

  const handleChange = (event) => {

    dispatch(setSendInfo({ sendMessage: event.target.value }));

  };

  // "0x4300000000000000000000000000000000000000000000000000000000000000"

  const handleClick = async (event) => {

    // console.log(await LoadSCpropReducer.contract.sendMessage())

    // console.log("sending" + ethers.utils.formatBytes32String(SendInfoReducer.sendMessage));

    // console.log("sending" + ethers.utils.parseBytes32String("0x6869696969696969000000000000000000000000000000000000000000000000"));

    console.log("sending");

    const result = await LoadSCpropReducer.contract

      .sendMessage(

        SendInfoReducer.toUserAddress,

        ethers.utils.formatBytes32String(SendInfoReducer.sendMessage)

      )

      .then(() => console.log("sent message"))

      .catch((e) => console.log(e));

  };

  return (

    <>

      <Grid item xs={12} sm={10}>

        <Box

          component="form"

          sx={{

            "& > :not(style)": { m: 1, width: "100%" },

          }}

          noValidate

          autoComplete="off"

        >

          <TextField

            id="outlined-basic"

            label="Message to Send"

            variant="outlined"

            value={SendInfoReducer.sendMessage}

            onChange={handleChange}

          />

        </Box>

40

</Grid>

      <Grid item xs={12} sm={2}>

        <Typography variant="h6" gutterBottom sx={{ mt: 2 }}>

          <Button variant="contained" disableElevation onClick={handleClick}>

            Send

          </Button>

        </Typography>

      </Grid>

    </>

  );

}

Handle Receive Message for an Account – ReceiveMessages/index.js

import \* as React from "react";

import Grid from "@mui/material/Grid";

import Button from "@mui/material/Button";

import Box from "@mui/material/Box";

import Typography from "@mui/material/Typography";

import { useSelector, useDispatch } from "react-redux";

import { setReceivedMessages } from "../../reduxProcess/actions/ReceiveMessageAction";

import RenderTable from "./renderTable";

export default function ReceiveMessages() {

  const { LoadSCpropReducer, ReceivedMessagesReducer } = useSelector(

    (state) => state

  );

  const dispatch = useDispatch();

  console.log(ReceivedMessagesReducer)

  const handleClick = async (event) => {

    console.log("Receiving");

    const result = await LoadSCpropReducer.contract

      .receiveMessages()

      .then(async (data) => {

        console.log(data);

        // content, timestamp, sender

        const result = await LoadSCpropReducer.contract.getMyInboxSize().then((size) => {

          console.log(size)

          dispatch(

41

setReceivedMessages({

              numSentMessages: parseInt(size[0].\_hex, 16),

              numReceivedMessages: parseInt(size[1].\_hex, 16),

              content: data[0],

              timestamp: data[1],

              sender: data[2],

            })

          );

        })

      })

      .catch((e) => console.log(e));

  };

  return (

    <>

      <Grid item xs={12} sm={10}>

        <Box

          component="form"

          sx={{

            "& > :not(style)": { m: 1, width: "100%" },

          }}

          noValidate

          autoComplete="off"

        >

          <RenderTable />

        </Box>

      </Grid>

      <Grid item xs={12} sm={2}>

        <Typography variant="h6" gutterBottom sx={{ mt: 2 }}>

          <Button variant="contained" disableElevation onClick={handleClick}>

            Refresh Received Messages

          </Button>

        </Typography>

      </Grid>

      {/\* <Grid item xs={12} sm={2}>

        <RenderTable />

      </Grid> \*/}

    </>

  );

}

42

Display the received messages in Tabular form – renderTable.js

import \* as React from 'react';

import Table from '@mui/material/Table';

import TableBody from '@mui/material/TableBody';

import TableCell from '@mui/material/TableCell';

import TableContainer from '@mui/material/TableContainer';

import TableHead from '@mui/material/TableHead';

import TableRow from '@mui/material/TableRow';

import Paper from '@mui/material/Paper';

import { useSelector } from "react-redux";

export default function RenderTable() {

    const { ReceivedMessagesReducer } = useSelector(

        (state) => state

      );

  return (

    <TableContainer component={Paper}>

      <Table sx={{ minWidth: 650 }} size="small" aria-label="a dense table">

        <TableHead>

          <TableRow>

            <TableCell>Sender (Address)</TableCell>

            <TableCell align="right">Content&nbsp;</TableCell>

            <TableCell align="right">Timestamp&nbsp;()</TableCell>

          </TableRow>

        </TableHead>

        <TableBody>

          {ReceivedMessagesReducer.messages.map((row) => (

            <TableRow

              key={row.content}

              sx={{ '&:last-child td, &:last-child th': { border: 0 } }}

            >

              <TableCell component="th" scope="row">{row.sender}</TableCell>

              <TableCell align="right">{row.content}</TableCell>

              <TableCell align="right">{row.timestamp}</TableCell>

            </TableRow>

          ))}

        </TableBody>

      </Table>

    </TableContainer>

  );

}

43

Contracts ABI file.

"abi": [

    {

      "inputs": [],

      "stateMutability": "nonpayable",

      "type": "constructor"

    },

    {

      "inputs": [],

      "name": "checkUserRegistration",

      "outputs": [

        {

          "internalType": "bool",

          "name": "",

          "type": "bool"

        }

      ],

      "stateMutability": "view",

      "type": "function"

    },

    {

      "inputs": [],

      "name": "clearInbox",

      "outputs": [],

      "stateMutability": "nonpayable",

      "type": "function"

    },

    {

      "inputs": [],

      "name": "getContractProperties",

      "outputs": [

        {

          "internalType": "address",

          "name": "",

          "type": "address"

        },

        {

          "internalType": "address[]",

          "name": "",

          "type": "address[]"

        }

      ],

      "stateMutability": "view",

44

"type": "function"

    },

    {

      "inputs": [],

      "name": "getMyInboxSize",

      "outputs": [

        {

          "internalType": "uint256",

          "name": "",

          "type": "uint256"

        },

        {

          "internalType": "uint256",

          "name": "",

          "type": "uint256"

        }

      ],

      "stateMutability": "view",

      "type": "function"

    },

    {

      "inputs": [],

      "name": "receiveMessages",

      "outputs": [

        {

          "internalType": "bytes32[16]",

          "name": "",

          "type": "bytes32[16]"

        },

        {

          "internalType": "uint256[]",

          "name": "",

          "type": "uint256[]"

        },

        {

          "internalType": "address[]",

          "name": "",

          "type": "address[]"

        }

      ],

      "stateMutability": "view",

      "type": "function"

    },

    {

45

"inputs": [],

      "name": "registerUser",

      "outputs": [],

      "stateMutability": "nonpayable",

      "type": "function"

    },

    {

      "inputs": [

        {

          "internalType": "address",

          "name": "\_receiver",

          "type": "address"

        },

        {

          "internalType": "bytes32",

          "name": "\_content",

          "type": "bytes32"

        }

      ],

      "name": "sendMessage",

      "outputs": [],

      "stateMutability": "nonpayable",

      "type": "function"

    }

  ],

The Script used for deploying the Smart Contract. (Scripts/deploy.js)

const hre = require("hardhat");

async function main() {

  const ChatWei = await hre.ethers.getContractFactory("ChatWei");

  const chatWei = await ChatWei.deploy();

  await chatWei.deployed();

  console.log("chatWei deployed to:", chatWei.address);

}

46

main()

  .then(() => process.exit(0))

  .catch((error) => {

    console.error(error);

    process.exit(1);

  });

Hardhat config file (hardhat.config.js)

require("@nomiclabs/hardhat-waffle");

// This is a sample Hardhat task. To learn how to create your own go to

// https://hardhat.org/guides/create-task.html

task("accounts", "Prints the list of accounts", async (taskArgs, hre) => {

  const accounts = await hre.ethers.getSigners();

  for (const account of accounts) {

    console.log(account.address);

  }

});

// You need to export an object to set up your config

// Go to https://hardhat.org/config/ to learn more

/\*\*

 \* @type import('hardhat/config').HardhatUserConfig

 \*/

module.exports = {

  solidity: "0.6.12",

};

GitHub Link: [Link](https://github.com/sachinthra/blockchain-chat-application.git) (https://github.com/sachinthra/blockchain-chat-application.git)

47

Some instruction to execute the project:

Use the below line of code to install required dependencies to run the Blockchain and smart contract.

npm install --save-dev @nomiclabs/hardhat-waffle ethereum-waffle chai @nomiclabs/hardhat-ethers ethers

To run the Hardhat server with 20 trial account, execute the below command in the command prompt from the root location for the project

npx hardhat node

To deploy the smart contract execute this below command in the CMD from root folder.

npx hardhat run --network localhost scripts/deploy.js

To Start the frontend, we need to move to the frontend folder so execute the below command in CMD from root.

cd frontend

Now to run the frontend execute the command in the same command prompt which is pointing to the frontend directory.

npm run dev

48

Chapter 4

**Conclusion & Future Work**

In the end of the project, A formal communication application based on blockchain technology is modelled as a decentralised chat application for sending formal chat messages using blockchain. This application can be used to verify availability, privacy and confidentiality of shared private data among clients of the organization. So here in this project we can see that the backend i.e., the blockchain is decentralised but the frontend is not a decentralised component as it need a server to host the frontend server.

In future I will remodel this application so that it can also exchange documents in the form of byte data. This feature will be lot more helpful for the organisations to share confidential documents which should ensure the privacy, security and immutability. So, in this project we have seen addresses are used to differentiate between the users but that won’t be optimal for a production application as it would be difficult for the users to find the receivers identity with the address. To overcome this problem, we can get the names and other details of the user at the time of registration and store it so that the names can be used to send and receive the messages.

49

**REFERENCE**

[1]. Secure Messaging Platform Based on Blockchain U. P. Ellewala; W.D.H.U Amarasena; H.V Sachini Lakmali; L.M.K Senanayaka; A.N. Senarathne IEEE 2020 2nd International Conference on Advancements in Computing (ICAC)

[2]. Secure Communications Using Blockchain Technology Peter Menegay; Jason Salyers; Griffin College MILCOM 2018 - 2018 IEEE

[3]. Secure Peer-to-Peer communication based on Blockchain Kahina Khacef, Guy Pujolle Publication: researchgate.net/publication/331767577

[4]. A Decentralized Connectivity Service for Private Social Communication Apps Christos Aslanoglou; Michalis Konstantopoulos; Nikos Chondros; Mema Roussopoulos 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)

[5]. A blockchain based secure communication framework for community interaction RahulSharmaa; MohammadWazida; ProsantaGope Journal of Information Security and Applications -Volume 58, May 2021, 10279027

[6]. BlockChat: A Decentralized Messenger on the Blockchain Andrew Zhang ;Sahil Shah ; Yassine Elmandjra ; Mark DeCesare ; Matthew Hanna https://fisher.wharton.upenn.edu/wp-ontent/uploads/2019/06/BlockChat.pdf

[7]. bChat: A Decentralized Chat Application Sourabh, Deepanker Rawat, Karan Kapkoti, Sourabh Aggarwal, Anshul Khanna International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 07 Issue: 05 | May 2020

[8]. Blockchain based secure communication application proposal: Cryptouch Recep Ahmet Sarıtekin; Eren Karabacak; Zübeyir Durgay; Enis Karaarslan IEEE 2018 6th International Symposium on Digital Forensic and Security (ISDFS)

[9] A new type of blockchain for secure message exchange in VANET. By RakeshShrestha; RojeenaBajracharya; Anish P.Shrestha; Seung Yeob Namb Digital Communications and Networks Volume 6, Issue 2, May 2020, Pages 177-186

[11] Zheng, Peilin, Zibin, Zheng, Weili, Chen, Jing, Bian, and Jianxun Eileen, Yang. "[EtherShare: Share Information in JointCloud Environment Using Blockchain-Based Smart Contracts](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.8705813&site=eds-live)." . In 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE) (pp. 233-2335).2019.

50

[12] B, Vardhini, Shreyas N, Dass, Sahana, R, and R., Chinnaiyan. "[A Blockchain based Electronic Medical Health Records Framework using Smart Contracts](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.9402689&site=eds-live)." . In 2021 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-4).2021.

[13] Arif, Yunifa Miftachul, Hani, Nurhayati, Sri, Harini, Supeno Mardi, Susiki Nugroho, and Mochamad, Hariadi. "[Decentralized Tourism Destinations Rating System Using 6AsTD Framework and Blockchain.](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.9079283&site=eds-live)" . In 2020 International Conference on Smart Technology and Applications (ICoSTA) (pp. 1-6).2020.

[14] Truong, Hien Thi Thu, Miguel, Almeida, Ghassan, Karame, and Claudio, Soriente. "[Towards Secure and Decentralized Sharing of IoT Data."](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.8946129&site=eds-live) . In 2019 IEEE International Conference on Blockchain (Blockchain) (pp. 176-183).2019.

[15] Epiphaniou, Gregory, Prashant, Pillai, Mirko, Bottarelli, Haider, Al-Khateeb, Mohammad, Hammoudesh, and Carsten, Maple. "[Electronic Regulation of Data Sharing and Processing Using Smart Ledger Technologies for Supply-Chain Security](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.8972565&site=eds-live)".IEEE Transactions on Engineering Management 67, no.4 (2020): 1059-1073.

[16] Das D, Banerjee S, Ghosh U, Biswas U, Bashir AK. [A decentralized vehicle anti-theft system using Blockchain and smart contracts. Peer-to-Peer Networking & Applications](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edb&AN=152012729&site=eds-live). 2021;14(5):2775-2788. Accessed February 12, 2022.

[17] Harshini, V M, Shreevani, Danai, H R, Usha, and Manjunath R, Kounte. "[Health Record Management through Blockchain Technology](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.8862594&site=eds-live)." . In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 1411-1415).2019.

[18] Omar, I. A., Hasan, H. R., Jayaraman, R., Salah, K., & Omar, M. (2021). [Implementing decentralized auctions using blockchain smart contracts](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edo&AN=150185957&site=eds-live). Technological Forecasting & Social Change, 168, N.PAG

[19] Shah, Meet, Mohammedhasan, Shaikh, Vishwajeet, Mishra, and Grinal, Tuscano. "[Decentralized Cloud Storage Using Blockchain](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85089991206&site=eds-live)." . In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184) (pp. 384-389).2020.

[20] Bandara, Eranga, Xueping, Liang, Peter, Foytik, Sachin, Shetty, and Kasun De, Zoysa. "[A Blockchain and Self-Sovereign Identity Empowered Digital Identity Platform](https://search.ebscohost.com.egateway.vit.ac.in/login.aspx?direct=true&db=edseee&AN=edseee.9522184&site=eds-live)." . In 2021 International Conference on Computer Communications and Networks (ICCCN) (pp. 1-7).2021.

51