FINAL REPORT

IFT 520

Secure decentralized Chat Application

Members:

Pallavi Dawn

Raunak Sengupta

Srijani Pal

C. GautamDev Chawdhary

INTRODUCTION:

History of Instant Messaging and Chat Systems is quite fascinating indeed. Starting off with telegram, telephony to Chats, Video and Text conferencing today is quite a leap. Chatting is mainly a part of Communication and Communication is a means to exchange dialogues whether verbal or via messages. Verbal means includes telecommunication which may include fixed lines to mobiles to wireless etc. Combine that with computer networks and you get applications like chat systems. There are two major principles of Communication:

1. **Client-Server**
2. **Peer-to-Peer**

As the names suggests, In Client -Server or centralized chat systems, there are multiple client nodes with a centralized server. Client and such web Server applications are pretty common nowadays. All information of user Login, authentication and transmit messages or data between customers is maintained by one server., Even User profiles.

While talking about security, it is clear that such a client-server application has its own prominent disadvantages, Is it secure? There is a huge overhead involved as all processing of such huge amounts of data is done by a Server, there can be connectivity issues which can down the signals between the client and server. Centralized systems can't be scaled after a certain limit even after increasing HW/SW abilities. On traffic spikes, since server only has a finite number of ports open, it can lead to Denial-Of-Service attack or DDoS attack.

WHY CENTRALIZED CHAT APPLICATION IS NOT SECURE?

One of the main drawbacks of a Centralized chat application is that a centralized storage of the existing users provides a single point of failure for attackers to compromise the security of the system. 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.

Such P2P decentralized systems have gained traction a lot now-a-days because it is more secure and scalable. In a decentralized P2P system, every node is independent and makes their own decision. All nodes act as peers and same priority is given to all nodes. Overall behavior of the system is dependent on the decisions made by each of this node.

Advantages of such a system: Minimal bottlenecks or performance issues as all the resources are divided between the nodes. There is high availability of the nodes as well, peers are always available. Some of the major applications are in Cryptocurrency like BitCoin or blockchain technology.

PROBLEM STATEMENT:

Our goal was to make a decentralized chat app in an attempt to create a more secure way of chatting. Most current applications, such as WhatsApp, operate on a centralized platform and therefore are more likely to be compromised.

This app will use Ethereum blockchain as a decentralized ecosystem (Hertig). Specifically, we will be using smart contracts to generate public keys for the end-users for security. This will be only accessible by the correct user.

The major problem with nowadays chat infrastructure is the presence of chat app, having a cost for sending fake messages would reduce people from sending messages to people. This app with the help the Ropsten Test Network helps us do that.

One more problem we are trying to solve is the privacy of people, just by having your phone number anyone is able to contact you on any of the centralised chat systems, here we are able to solve this problem by using blockchain and having cost for the same.

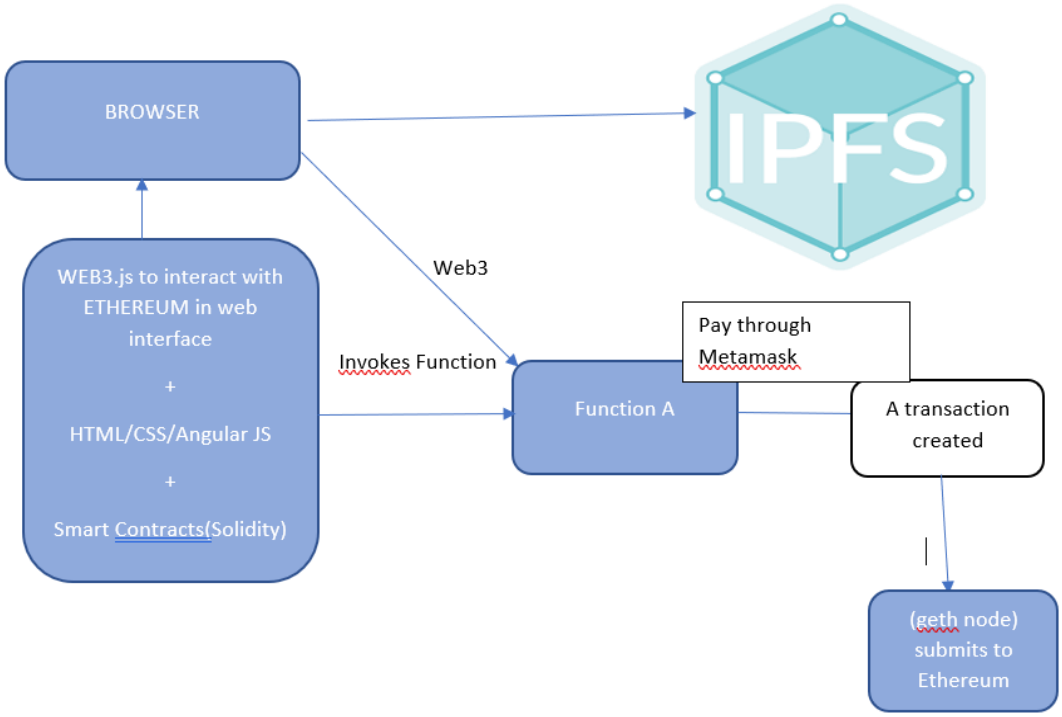
OBJECTIVES:

The primary motivation for implementing this project is to design a much more secured application which can leverage the issues of a centralized server. Thus, the advantages of a distributed chat application over a centralized server are: -

- No single server resulting in less overload / attacks
- No single point of failure
- Scalable
- Tackling the fake messages problem
- Data Immutability

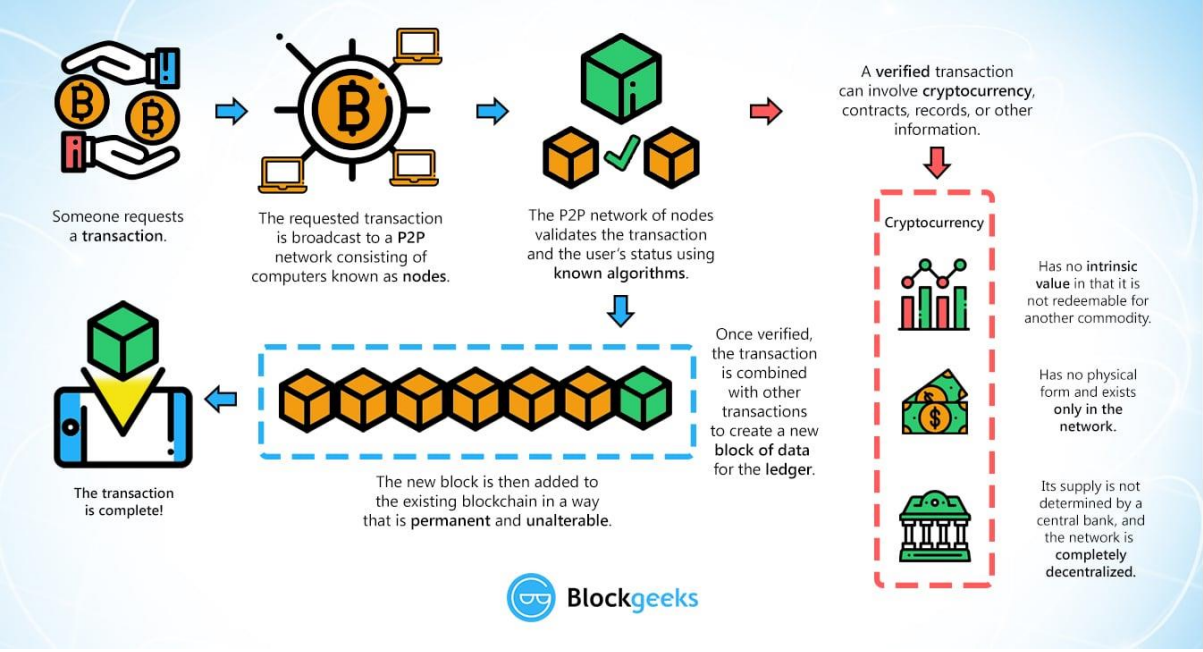
Also using smart contracts and the extremely secured Ethereum blockchain there is very less chance of data stream being compromised with fellow peers. Also, credible transactions are made through the smart contracts that prevents involvement of the third parties.

GRAPHICAL ARCHITECTURE OF THE APPLICATION:



COMPONENTS DESCRIPTION:

- **Blockchain:** It is a distributed ledger, consisting of immutable set of data records which is managed not only by one machine but by multiple machines. It has no central authority. Our project is based on the concept of blockchain. The meta data is there for anyone to see . It heeds transparency.



- **Ethereum:** It is a framework basically where developer can built web applications on top of blockchain. It gives greater power to developer to create DAPPS. Here, miners are there who compete to get ethers to verify transactions. Difference between bitcoin and Ethereum is that bitcoin basically is a cost system but Ethereum is more of a platform to run applications while paying for transactions.

- **Web3 provider:** It is like a wrapper to have all Ethereum codes. When a user renders a screen or a page, Metamask injects a etheruem provider and a web3 instance which the screen utilizes, then only there is a connection between Ethereum and web interface.

- **Metamask:** It is a wallet where you can pay for all the transactions you make**.**

- **Smart Contracts:** It is a set of strict rules or protocols that has be followed by users who use the contract. They have to abide by this rules. any transactions that happen on the blockchain happens because of the functions mentioned in the smart contract code. This creates more transparency , clarity of how much you have to pay for each transaction and what are the requirements.

- **Angular JS**: It is a front-end Javascript framework managed by Google, open source and extends the HTML features with directives.

- **IPFS:** It is a file hosting system where you can host you files and share it will multiple peers once you get the hash address to run it on ipfs.io

STEPS INVOLVED:

- We build the entire project folder on Truffle framework and create smart contracts for the Chat functionality on Solidity programming language.

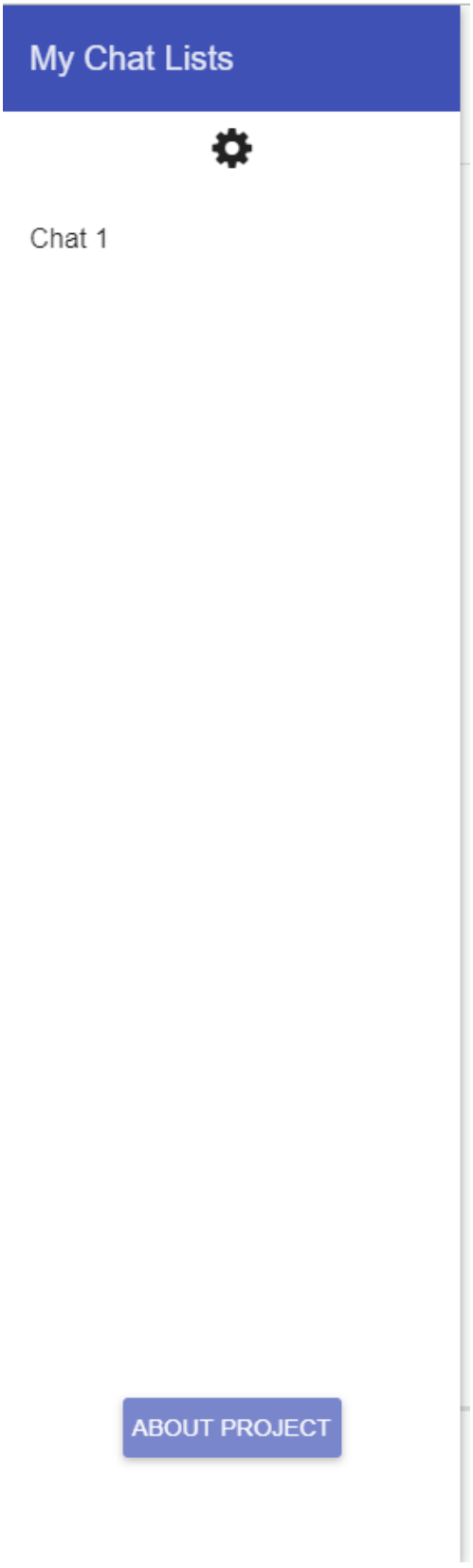
- We use Web3.js which is a library file to retrieve all the Metamask extension parameters and eventually interact with the Ethereum blockchain from the browser using Metamask.

- A sender and a receiver who are the end users invokes the chat functionalities using the smart contracts. The main functions implemented on smart contracts are: -
    1. SendChatMessages()
    2. CreateNewUser()
    3. GetTotalCountMessages()
    4. ReceiveChatMessages()
    5. Kill()

- After getting the Ropsten API Key, we then configure the Ropsten test network and deploy the above contracts on it.

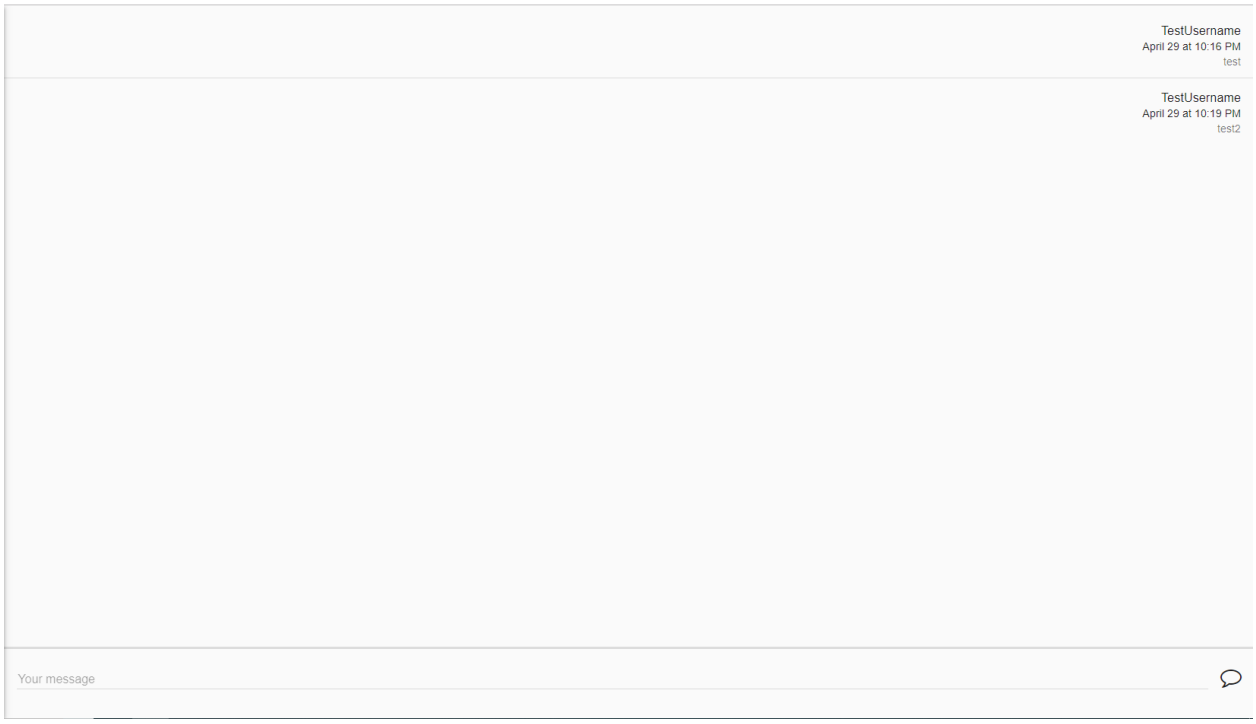
- The sender and receiver has to have a Metamask wallet so that they can pay through real ethers to make transactions and submit it on Ethereum and subsequently use the chat app.

SCREENSHOTS AND CORRESPONDING CODES:

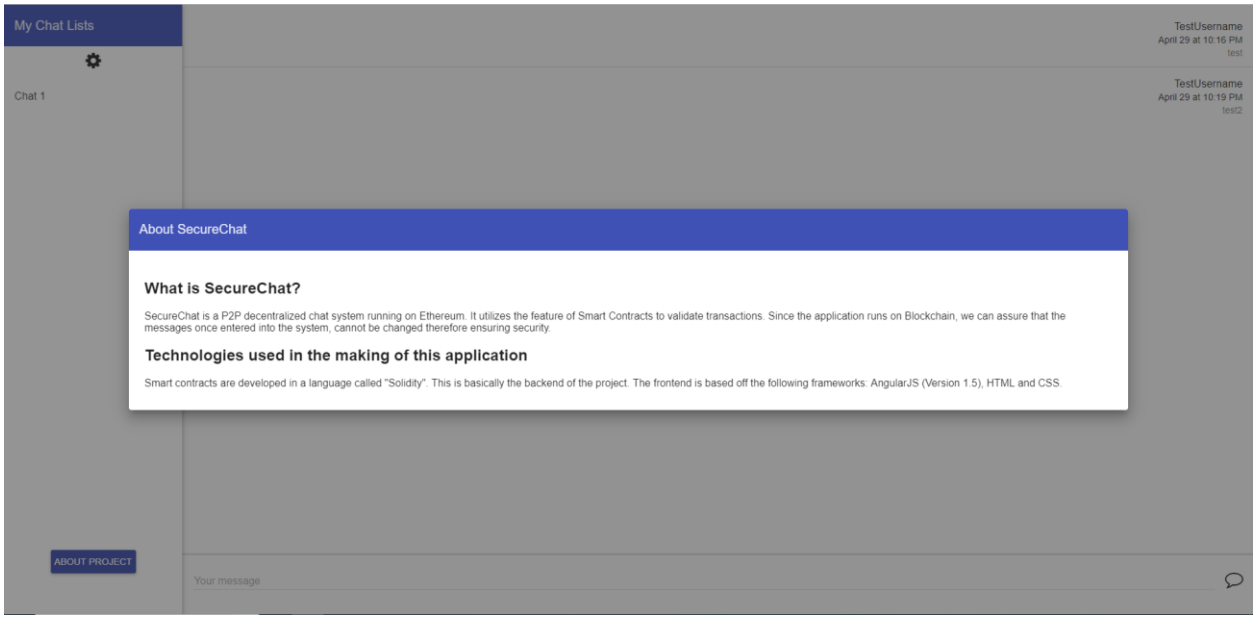


This screen shows us the list of all the chats you are currently having.

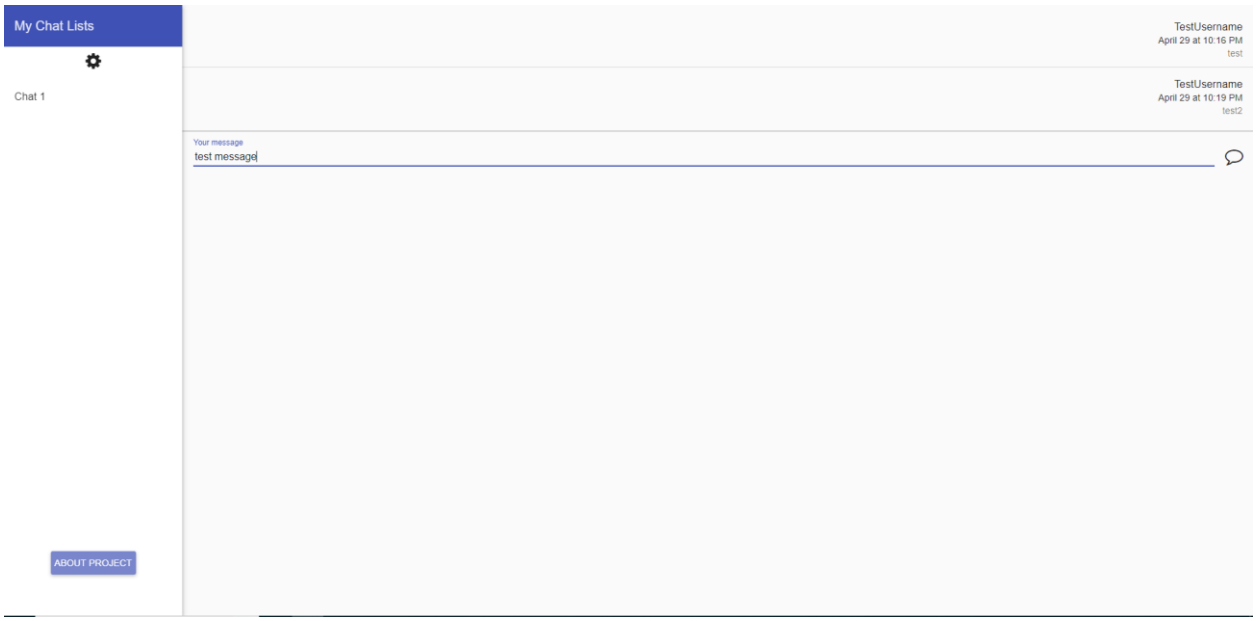
```
</div>
```



This window is chat window, where you can chat with the other user.

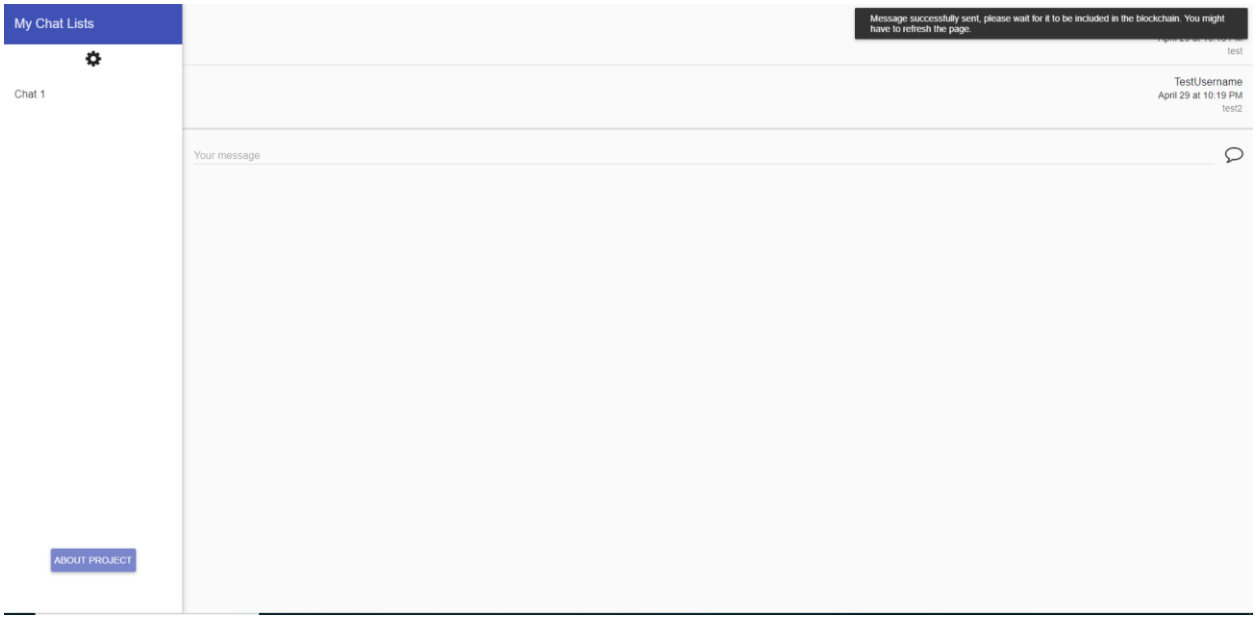


This screen is what happens when you click on the about SecureChat button on the app.

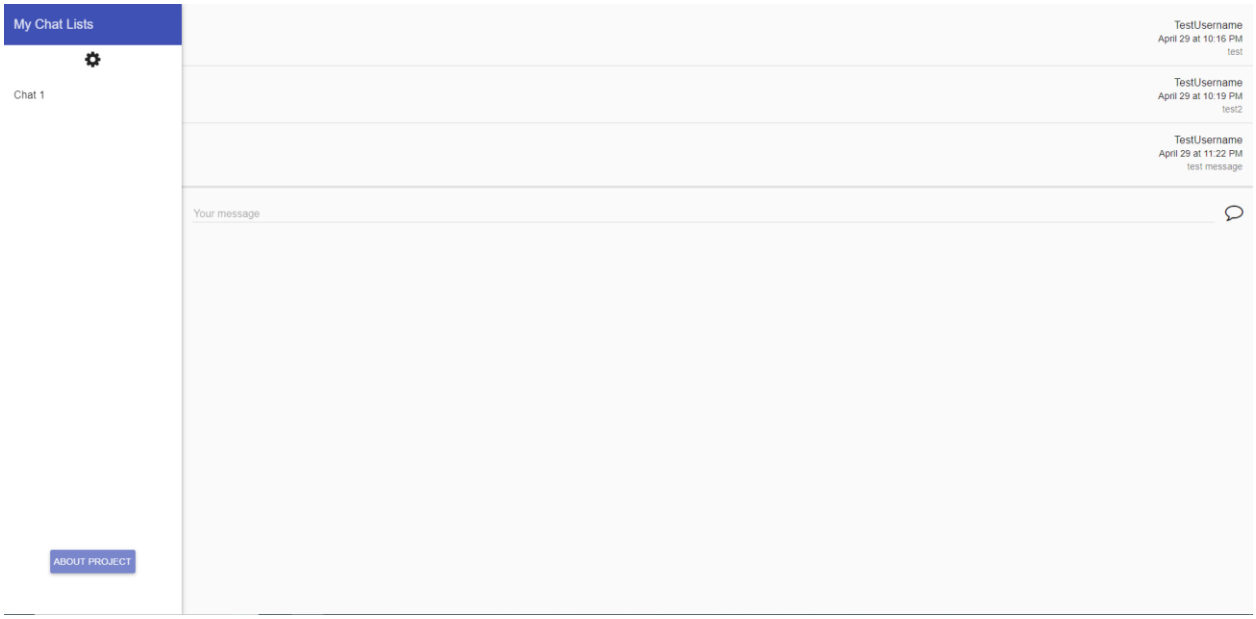


This window shows us the message received from the other user

This window shows the charge on every message that goes through the app



This window shows the message being generated on successfully sending the messages



Main Code:

SECCHAT.sol

We are using the 5.07 version of solidity. It's the language provided by Ethereum.

```
pragma solidity >=0.4.21 <0.6.0;

contract SecChat {

    event NewMessage(string message_data, address userAddress, uint timestamp, string roomName);

    address payable owner;

    struct Messages {
            string message_data;
            address userAddress;
            uint timestamp;
        }

    mapping(string => Messages[]) roomMessages;
    mapping(address => string) addressToUsername;

    function getMessageCountForRoom(string calldata _roomName) external view returns (uint) {
            return roomMessages[_roomName].length;
        }

    function kill() external {
            if (owner == msg.sender) {
                selfdestruct(owner);
            }
```

```
    }

  function sendMessage(string calldata _msg, string calldata _roomName) external {
        Messages memory message_data = Messages(_msg, msg.sender, block.timestamp);
        roomMessages[_roomName].push(message_data);
        emit NewMessage(_msg, msg.sender, block.timestamp, _roomName);
    }

  function getMessageByIndexForRoom(string calldata _roomName, uint _index) external view returns (string
memory, address, uint) {
        Messages memory message_data = roomMessages[_roomName][_index];
        return (message_data.message_data, message_data.userAddress, message_data.timestamp);
    }


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

  function createUser(string calldata _userName) external {
    addressToUsername[msg.sender] = _userName;
  }

  function getUsernameForAddress(address _userAddress) external view returns (string memory) {
    return addressToUsername[_userAddress];
  }

}
```

CODE:

We have included the code as zip file in the drive link. This code has the smart contracts written in solidity. This also has frontend written in angular JS. Whatever could we could show we have included in the above code.

FUTURE SCOPE:

- Utilization of peer to peer network to show DDOS attack.
- Making the contract more full proof with more functions like involving one on one architecture only.
- Encryption and decryption of the messages.
- More counter measures to make sure the peers are not bad.
- Create an android version of the same concept that we used here.

CONCLUSION:

We thereby have created a very secure Chat app which has encountered the problems faced by the current centralised Chat Apps and also the decentralised Chat app by addressing problems related to privacy, scalability, security, dependability.

This could in future be the way to communicate as soon as make verification of message faster, which is the current problem being researched by various blockchain based companies.

We could in future extend this to Chat app where in the user can be verified before you could communicate with them. This could help in interview people, meeting new people and variety of other applications where the security and privacies of the people of both the user and the client side can be maintained.

REFERENCES:

1. [Far18] Carson Farmer. The definitive guide to publishing content on the decentralized web, https://medium.com/textileio/the-definitive-guide-to-publishing-content-on-ipfs-ipns-dfe751f1e8d0, Aug 2018. [Last accessed: 30 Apr, 2019]
2. [Anu17] Anurag Srivastava Interacting with Smart Contracts using Solidity & Web3, https://medium.com/zeonlab-blockchain-semantic-blog/interaction-with-solidity-using-web3-5a4f1a7166f3, Dec 14. [Last accessed: 30 Apr, 2019]
3. [Open18] Blockchain-Based Distributed Protocol Open Chat White Paper, http://www.openchat.co/whitepaper/OpenChat_zz.pdf, Mar 18. [Last accessed: 30 Apr, 2019]
4. [Aly18] How Do Ethereum Smart Contracts Work? https://www.coindesk.com/information/ethereum-smart-contracts-work, Jul 18, Last accessed: [Last accessed: 30 Apr, 2019]
5. [Mer17] The ultimate end-to-end tutorial to create and deploy a fully decentralized Dapp in ethereum, https://medium.com/ethereum-developers/the-ultimate-end-to-end-tutorial-to-create-and-deploy-a-fully-descentralized-dapp-in-ethereum-18f0cf6d7e0e, Aug 17. [Last accessed: 30 Apr, 2019]