DECENTRALISED CHAT APPLICATION

Uma Thakur[1], Abhishek Chichmalkar[2], Aditya Sambhare[3], Aman Chaturvedi[4], Chinmay Khuspare[5], Nikhil Tembhe[6]

[#]Department of Computer Science and Engineering, Priyadarshini College of Engineering, Nagpur, Maharashtra, India

---***---

Abstract - When we wanted to buy a product like a watch on the internet, then suddenly we see ads related to the product on Facebook, Instagram, Google, etc this means they are accessing our every information even the messages we send to people on the Internet. There is a growing suspicion that the web is betraying and spying on us. As communication is an important part of an individual's lifestyle, as each person communicates globally with the means of the internet every day and as in today's world, different chat systems are almost working on centralized systems i.e., all the data is in a centralized server. Therefore, a major problem is if the central server fails then the whole network fails, and due to this major drawback is that there can be a loss of user's data, information, and resources which is stored on the centralized server or even there can be a leak of user's chat information that is stored on the server.

Decentralized is the way to resolve this problem, it's an internet hosted via a peer-to-peer network. The information will be distributed and stored around the world on multiple devices like phones, laptops, and even smart appliances. To achieve this, we are using Gun.js, a decentralized graph database that is real-time has Low latency, and also has security, encryption, and authorization for the data.

Key Words: Gun.js, Radisk, webRTC, Svelte, Decentralized server, User Interface, Peer to peer network.

1. INTRODUCTION

In today's world chatting over the messaging platforms are a part of an individual's lifestyle. The most of the communication whether it is a confidential chat or normal chat are mostly happens over in social media platform. As we all know that most of the traditional chat application are working on a centralized server i.e., all the user data is stored on a central server. So, the major problem of this server is if the central server fails then the whole data collapses (break down) or even can leak the user information stored on the server. This is the major drawback of this system, but in today's world privacy makes a very important role in everyone's life and specially for those organizations that works on confidential tasks like for example the military purpose, they need a very secured privacy to handle their communication from the one end to the other end. So, to achieve this system our project had made the use of decentralized backend server, as the name suggests decentralized server it does not have any central server. It is peer to peer network i.e., all the computers are linked together with equal permissions for processing data. To achieve this decentralized backend server, we used a library called as Gun.js, a decentralized graph database. In this, for front end UI of our chat application, we used Svelte a JavaScript library. Decentralized application consists of multiple nodes connected to each other in a mesh type of topology network. It's just like a peer-to-peer network that data is stored in such a way that it is almost impossible to view by an anonymous user.

2. PROBLEM STATEMENT

A. Data is stored in centralized server.

As to storing the user data, it has been stored in big servers as it also takes an ample amount of space. It has been said that if any one of the servers fails, the whole network collapse due to that loss of data, which is dangerous for the user as if their data has been lost. Due to that, you cannot retrieve your information back to overcome this issue.

So here comes our project in the picture, whereas the high-tech company stores their user in the server. At the same time, there is no guarantee that your data will be safe. Where data is stored in a hard disk somewhere in the cloud, which has been shared across multiple machines, whereas our project stores a small subset of data on each user based on the actual data that they consume via a peer-to-peer connection.

B. Lack of loyalty

As in today's fast-growing world, whenever we search for some product in a search engine after a while, we see a pop-up coming out of know ware showing the exact related product that you searched a while ago. Guess how they came to know that you were looking for this product. This may lead someone is monetizing your activity that your data is not secure, which means your data has been shared with a different company that you are not aware of, which means you have no longer control over your data where as in our project the data is stored in a subset of data based on actual data. Which is much more secure than the centralized server.

3. COMPONENTS

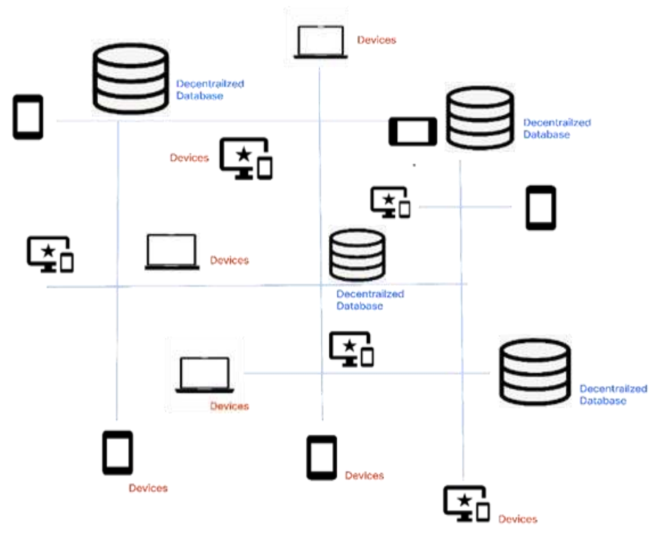

Fig.1: Diagram of Decentralized web

Gun.js: -
Gun.js decentralized graph database is a centralized database that which on a server hosted by big tech data in the gun is distributed across multiple peers or users through the use of webRTC and end-user like user requires a specific graphic data to use your app while another user requires an entirely different one with a few shared data points in between each peer can store the data it needs in the browsers local storage then sync changes with other peers in the network a peer might be one of your end-users or a relay server that you deploy to make the system more reliable that means the entire database is the union of all graphs on the network, and no one machine controls the entire system a decentralized fire stored where you get real-time updates offline mode and the illusion of what latency the data itself is managed through a minimal graph API where each record is a node that contains a unique ID and any other custom data for that entity in plain JavaScript any node on the graph can reference another node allowing you to create an infinite chain of circular dependencies. an individual node never contains a duplicate of another node just a reference to it allowing you to model complex relationships without the need for a schema.

Radisk: -
The Radisk Storage Engine (RAD) is an in-memory, as well as on-disk radix tree that saves the GUN database graph for fast and performant look-ups. Radix trees have a constant lookup-time.

webRTC: -
webRTC exchange real-time audio video streams with your friends entirely in the browser to capitalize on the work from home boom then webRTC is the API because it allows you to establish a peer-to-peer connection between two or more browsers where they can exchange audio video media directly without the need for a third-party server or native app webRTC will make a series of requests to a stun server.

Svelte: -
Svelte it's a JavaScript tool for building UI components, whereas svelte is a compiler it converts the declarative code that writes as a developer into imperative code that works with native browser APIs. As a result, to get highly performant code in a tiny package, but most importantly, it's the only JavaScript framework that's actually used to create components in dot spelt files which contain three main parts a script for your JavaScript code which can also be typescript a style tag for your CSS which can also use a preprocessor like as and the main template represented as HTML

4. PROJECT DESCRIPTION

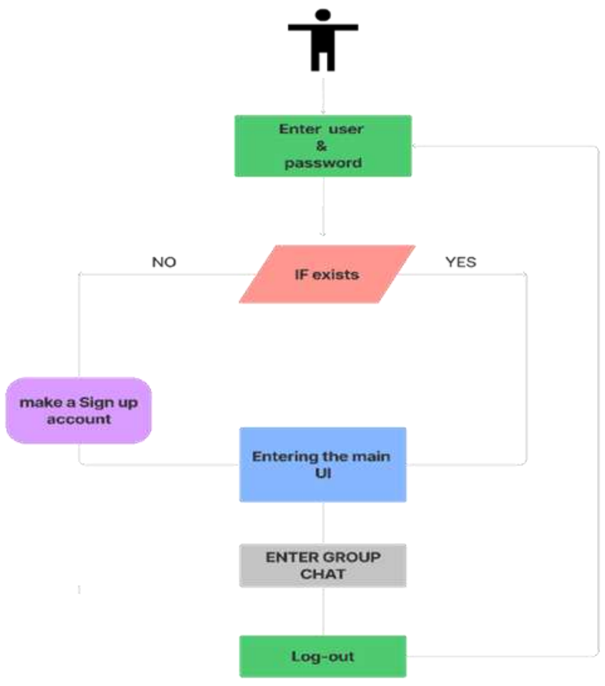

Fig. 2: User Flow Diagram

We made a chat web app where the data and infrastructure are not controlled by a big tech company instead it is decentralized across the entire user base using web technologies. We built this decentralized chat application using a library called gun.js and for the front-end UI, we have used Svelte a JavaScript library (an open-source front-end compiler).

A gun.js is a decentralized graph database, a normal database that stores all the data in a hard disk somewhere in the cloud it may be shared across multiple machines, but for all intents a purpose we can imagine our entire data set on a single disk whereas gun.js also stores a small subset of data on each user based on the actual data they consume in the application.

When a user makes a query for some data it will search across the network for the other users that have that data and sync up using webRTC which is a web real-time communication, so it is like an entire database as the union of all peers on the network. It's like a peer-to-peer network that data is stored in such a way that it is almost impossible to view as a very secure encryption function is used, this idea is very similar to a blockchain ledger where no individual has control of the entire network but it is not a blockchain technology which tends to be too slow.

For user authentication when we create a new user account this will generate secure key pair cryptographically and the key is associated with the public key so the past message can be found and the password is proof of work seed to prove and to decrypt access to an account's private keys.

For the data to be persisted if a user just clears the browser cache data, in the browser's the local storage is limited to 5 megabytes and if a user clears out the data in local storage it could potentially be lost if it's not somewhere else on the network probably not acceptable but for this, we deploy a relay server that uses a different storage mechanism called Radisk that can store a lot more data on the disk of an actual server and which will make the network more robust because if a query falls back to a relay if it's not available from another peer by creating a node in the database which will be available to the entire decentralized network.

5. PROJECT IMPLEMENTATION

We have used a Svelte app with two main features email password user authentication and then a massive group chat room that anybody can join the only dependency in this project is gun.js which we installed with npm.

The first thing we will focus on is user authentication and to do that we created a file called user.js, at the file we imported gun.js, and below that we imported two supporting libraries the first one SEA (Security Encryption and Authorization) which is the module that enables user authentication and we use AXE (Advanced Exchange Equation) it is an alternative way to connect peers together and tends to be more performant for a chat app.

We created a variable called DB to initialize the database then used it to refer to the currently authenticated user, we also changed the option of recall to session storage, so the user stays logged in between browser sessions.

To manage the user in the app, we need to know if the user is logged in and we also want their username so we can show it in the UI (User Interface). We get the username by making a reference to the user then calling git alias which will be the value of whatever username users choose and when they sign up, we then going to use this value frequently throughout the application.

So, for that, we imported writable from the Svelte store to make it reactive. A store is like an observable value that will re-render the UI whenever it's changed and it can be shared across multiple components so whenever the alias for the current user updates, we will set the value of the store to that value but we also want to listen to changes to the authorization state when the user signs in sign-out.

To handle that we'll listen to the authorization events on the database and do the same thing where we fetch the alias and then set that as the value on the store.
In Svelte we have the Header - it's component where it will show the title of the app when you're not logged in but if you are logged in it'll show your username as well as your avatar.

Now to implement sign out we imported the username store and the gun user object then we created a function called sign out that calls will leave on the user and set the username store to an empty string. Then in our HTML file, we have a button that when clicked we'll call the sign out a function but we only want to show that button if the user is signed in and the way we can tell if a user is signed in is if their username is present in svelte, we do that with a store in any component by adding an if statement followed by a '$' dollar sign and then the name of the store. The dollar sign will subscribe to the store and react to any changes that happen to it if the username does exist then we will go ahead and show the username by subscribing to the store once again, we also did the same thing to show an avatar with the user's initials by using DiceBear API which will make a unique avatar based on the username as the random seed, then we have a button that has the sign out function when clicked and we add an else block to show something else if the user is not logged in, this is about the header component and we declared it in the app component also.

Now on to the login component which will allow the user to sign in or sign up with their username and password, we will import the user object and then set up the local state for the username and password.

In svelte we can implement two-way data binding with a variable by creating a form input and then stating bind value to that variable which means anytime user types into the form, the value of the variable will change. We have created separate form inputs for username and password, and we will share them for both the login and signup process which itself just consists of two different buttons that will either call the login function or the sign-up function.

To user log in, we used the user auth method which consists of three arguments, takes the username and password as first and second arguments and in the third argument, we defined a callback, in this case the callback is the value of the store that we set up earlier which will update automatically.
Now to sign up a new user is very similar to the previous but instead of calling auth we call that create method on the user and if the user creation is successful then we will log that user in automatically.
Now at this point, we have a working user authentication system.

For the main chat component which is where we query items from the database and which also gives the user the ability to send a message.

The chat component has two pieces of state, a string for a new message that the user will type into form input and an array of messages which will contain the message text along with the user who sent it and a timestamp.
To query messages, we set up the onMount lifecycle hook that will run whenever the component is first initialized, inside it, we used the database to make a reference to the chat node as for creating super chat we just give it a name of chat.
Then we just call map to loop over every single message in the chat and then call once to only read each message once, in this case each message is immutable so we don't need to listen to changes in real-time, then we defined a call-back that will be called on each new message that will give us access to the data and the id of that node.

Now if the data is defined, we format a message more suitable for the User Interface with the properties of who, what, and when.

1. Who - It is the user that sent the message and we can figure out who sent it by taking that data of the raw message and using it to get the alias,
2. What - It is the actual text of the user's message,
3. When - It is the property to get an accurate timestamp across all of our users, we're using Gun state as the final source of truth with the raw data, this gives us all the data we need for the User Interface.

Then we take each message and add it to an array that we can then loop over it in the UI (User Interface) with Svelte. We loop overran array of items in svelte with each this will render the chat message component for each message in the array and by adding parentheses with message-When, which will provide svelte with a unique key to sort all the messages efficiently by which we read messages in the chat.

We made a form to submit them in the UI (User Interface), in the form on the submit event will call the send message function. Inside that function, we use SEA to encrypt the actual message text and the key is the same as the one we used earlier to decrypt the messages. Now we can associate the message to the current user using the encrypted message as the value then we created a date to serve as the index for the message so it can be sorted properly at which point, we can reference the chat collection create a new node based on that index and store the message value.

6. ADVANTAGES

1. For Government/Military officials: - It will be very useful for government security officials when they need to send their confidential messages or information regarding any work or to send their data to their intenders for some official reasons without any third-party involvement. As they don't need to trust any other application.

2. Users don't have to put trust in any Central authority.
    a. You don't need to trust any other chat application that is working on a centralized server or followed by the third parties that use or sell your data, information, photos to any adman who follows you around the internet. As this decentralized chat application will be useful for securing your data you should be able to reduce or eliminate the trust that you're required to put into third parties.

7. CONCLUSION

Decentralized applications tend to form the interaction between two people more efficient and simpler. The chatting process nowadays features a mediating node, while our software doesn't have any mediating device/node i.e. Every person is connected by a peer-to-peer network. A Decentralized network can be a key technology that can solve privacy and confidentiality-related issues that exist in the Traditional or existing messaging system.

8. REFERENCES

[1] Sourabh, Deepanker Rawat,Karan Kapkoti,Sourabh Aggarwal,Anshul Khanna, "bChat: A Decentralized Chat Application",Inderprastha Engineering College, Uttar Pradesh, India,(IRJET).

[2] Peter Menegay, Jason Salyers,Griffin College,"Secure Communications Using Blockchain Technology",Milcom 2018 Track 3 - Cyber Security and Trusted Computing.

[3] Suman Kumar Das, Zenlabs, Zensar Technologies, Pune, India,"Secure Messaging Platform Using Blockchain Technology", (IJRES), ISSN (Online): 2320-9364, ISSN (Print): 2320-9356.

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

[5] Muhammed Kuliya, Hassan Abubakar,"Secured Chatting System Using Cryptography", Lovely Professional University, Punjab India,(IJCRT).

[6] "A WebRTC Video Chat Implementation Within the Yioop Search Engine"(2019). By Ho, Yangcha, Master's projects. 726. DOI:https://doi.org/10.31979/etd.dtz2-hstt .

[7] "A secure Chat Application Based on Peer-to-Peer Architecture", Mohamad Afendee Mohamed, Abdullah Muhammed, and Mustafa Man on 28-05-2015 in Journal of Computer Science.

[8] Secure Peer-to-Peer communication based on Blockchain Kahina Khacef, Guy Pujolle.

[9] https://gun.eco/docs/Radisk

[10] https://gun.eco/docs/API

[11] https://webrtc.org/getting-started/peer-connections

[12] https://svelte.dev/docs

[13] https://relay.dev/docs/