Final Project: Hybrid centralized and p2p chat system SOCKET

Distributed System Group 4

16/4/2020

Contents

1 Introduction

Like many great chefs who like to cook their own foods. We as a programmers, who also likes to build our own application, because we can have absolute control to add and remove things we like and dislike while also have better privacy.

Although we cannot replace anything and everything, and we also don't have the knowledge or resource required to do so alone. We could, however, build a simple small-scaled application for the sake of gaining experience and deeper understanding of the technology rather than to replace what other people have made very well.

So with the help of Dr Son and what we have learned in the Distributed System course, we took on the challenge to make a p2p chat system, and in this report we will detain How we design and build the system, What tools did we use, And what we have accomplished.

2 Model

Basic flow When a new client connects to the server. The server will tell existing clients to connect with the new client, forming p2p connection. Also, same thing applied when the server tells other clients that a client has disconnected, or terminated.

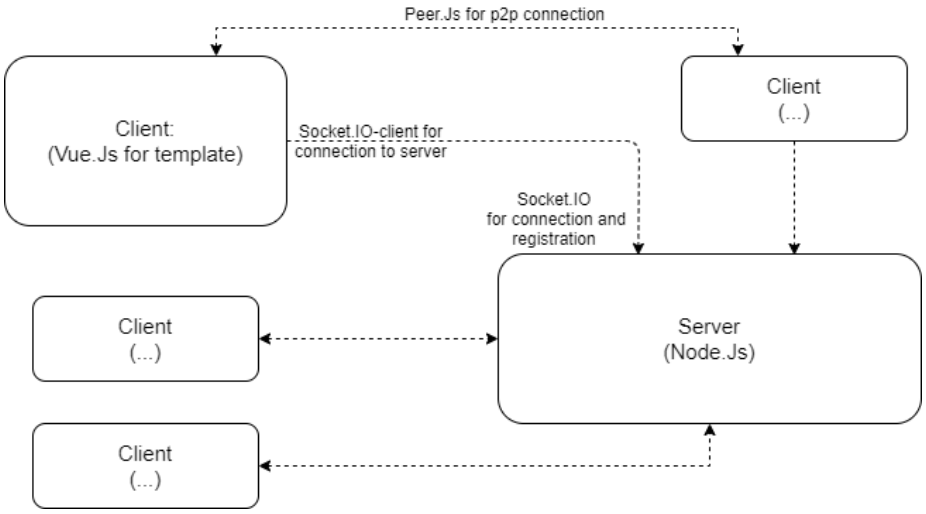

Figure 1: Model

3 Tools

3.1 PeerJS

Definition PeerJS simplifies peer-to-peer data, video, and audio calls.

Usage Every Peer object is assigned a random, unique ID when it's created.

```
    peer.on('open', function(id) {
    console.log('My peer ID is: ' + id);
    });
```

When we want to connect to another peer, we'll need to know their peer id. You're in charge of communicating the peer IDs between users of your site.

Data connection Start a data connection by calling peer.connect with the peer ID of the destination peer.

```
        var conn = peer.connect('dest-peer-id');
```

Listing 1: Start connection

```
        peer.on('connection', function(conn) { ... });
```

Listing 2: Receive connection

Anytime another peer attempts to connect to your peer ID, you'll receive a connection event. peer.connect and the callback of the connection event will both provide a DataConnection object. This object will allow you to send and receive data:

```
        conn.on('open', function() {
            // Receive messages
            conn.on('data', function(data) {
            console.log('Received', data);
            });

            // Send messages
            conn.send('Hello!');
        });
```

What kind of data can I send? PeerJS has the BinaryPack serialization format built-in. This means you can send any JSON type as well as binary Blobs and ArrayBuffers. Simply send arbitrary data and you'll get it out the other side:

```
        conn.send({
            strings: 'hi!',
            numbers: 150,
            arrays: [1,2,3],
            evenBinary: new Blob([1,2,3]),
            andMore: {bool: true}
          });
```

What about latency/bandwidth? Data sent between the two peers do not touch any other servers, so the connection speed is limited only by the upload and download rates of the two peers. This also means you don't have the additional latency of an intermediary server.
The latency to establish a connection can be split into two components: the brokering of data and the identification of clients. PeerJS has been designed to minimize the time you spend in these two areas. For brokering, data is sent through an XHR streaming request before a WebSocket connection is established, then through WebSockets. For client identification, we provide you the ability to pass in your own peer IDs, thus eliminating the RTT for retrieving an ID from the server.

3.2 SocketIO

Definition Socket.IO is a library that enables real-time, bidirectional and event-based communication between the browser and the server. It consists of:

- a Node.js server
- a Javascript client library for the browser (which can be also run from Node.js)

Its main features are:

Reliability Connections are established even in the presence of: proxies and load balancers. personal firewall and antivirus software. For this purpose, it relies on Engine.IO, which first establishes a long-polling connection, then tries to upgrade to better transports that are "tested" on the side, like WebSocket. Please see the Goals section for more information.

3.3 VueJS

Vue is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

4 Working Method

4.1 How server work

We build our own server base on Nodejs and socketIO, which help us to listen the event on client.

```
        socket.on('username',username => {
            OnlineUser.push(new User(username,socket.id))
            console.log(OnlineUser);
            setTimeout(() => socket.broadcast.emit("join_server",
                username),0);
        });
```

This code make the server listen to the "username" event, which is submit the ID of the user to the server, also save it to database.
After that, another listenner also applied to listen for "disconnect" event, which to know when the user disconnected from the server then remove that user in database.

```
        socket.on('disconnect', () => {
        let index = OnlineUser.findIndex(user => user.id == socket.
            id);
        if(index != -1){
            socket.broadcast.emit("leave_server",OnlineUser[index].
                username);
            OnlineUser.splice(index,1);
        }
    });
```

4.2 How client work

We use vue-peerjs and vue-socket.io-extended, which is the adaptive package that community develop to make a better experience when working with vuejs, two those packages still are peerjs and socket.io-client.

```
            import VuePeerJS from 'vue-peerjs';
            import VueSocketIOExt from 'vue-socket.io-extended';
```

Then we listen to peerjs Implementation

```
            peer_open({dispatch,commit},id){
                id = getSession();
                this._vm.$peer.on('connection',function(conn){
                    let chatZone = new ChatZone(conn);
                    conn.on('data',function(message)  {
                    dispatch('addMessage',{username:conn.peer,
                        message})
                    })
                    commit('addConnection',chatZone);
                })
            },
            peer_close(){
                console.log('Connection destroyed');
            },
            peer_error({commit},err){
                console.log(err);
            }
```

When each user is initialized, this client will send "username" event to server through SocketIO

```
            this.$socket.client.emit('username',getSession());
```

Note that getSession() here is the last username on that tab/client.
And also listenner to server event

```
            socket_leaveServer({commit},username){
                commit('removeConnection',username);
            },
            socket_joinServer({dispatch,commit},username){
                let conn = this._vm.$peer.connect(username);
                let chatZone = new ChatZone(conn);
                conn.on('data', function(message) {
                    dispatch('addMessage',{username,message})
                })
                commit('addConnection',chatZone);
            },
```

5 Conclusion

Hybrid centralized and p2p chatsystem chat system is very usable and dynamic. This system run with low latency and bandwidth because it only have to contain the user infomation, instead of all the message. P2P chatsystem meaning more security because the message can not read by the server, only client know those messages.

6 Who does what?

- Bui Quang Huy: Build the main concept of the system and how they work.
- Nguyen Viet Dung: Research on client side development, design client methods and events.
- Nguyen Quang Trung: Design client concept and interface.
- Do Minh Hoang: Research on server side development
- Pham Minh Kien: Design server methods and events.