Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 3 - Chat Multicast

Unidad de aprendizaje: Aplicaciones para Comunicaciones de Red

Grupo: 3CM8

Alumnos(a):
Nicolás Sayago Abigail
Ramos Díaz Enrique

Profesor(a):
Moreno Cervantes Axel

30 de Abril 2019

Índice

1. Introducción

En ésta práctica de implementa una aplicación que brinda un servicio de chat multiusuario con la siguiente funcionalidad:

- Sala de chat comunal
- Envió de mensajes de texto, imágenes, emojis predefinidos, y archivos
- Envió de mensajes privados

Se utilizo el lenguaje de programación JavaScript, con ayuda de ciertas bibliotecas y frameworks como JQuery, SocketIO que utiliza sockets de flujo bloqueantes implementados con hilos (a diferencia de Java, que utiliza sockets de datagrama), Validetta, etc. por lo que es una aplicación web.

2. Desarrollo

2.1. Instalando SocketIO

Antes que nada, se instalaron las herramientas y el framework de SocketIO para el desarrollo de esta práctica:

```
sudo apt-get install nodejs
npm install --save express@4.15.2
npm install --save socket.io
npm install --save socketio-file-upload
```

Toda la documentación del framework esta disponible en su página web: `https://socket.io/`

2.2. Configuración del Servidor

Primero que nada, se configura el servidor para que escuche peticiones de clientes que se conecten a él, utilizando el puerto 3003.

Le indicamos que vamos a utilizar el framework SocketIO junto a su módulo para manejo de archivos, qué ruta será el index de toda la aplicación en general, así como los archivos HTML que serán las vistas para el usuario.

El bloque *io.on("connection")* indica que un socket cliente se ha conectado (similar al ciclo infinito en el lenguaje Java), atendiendo peticiones infinitamente.

Para enviar, se utiliza el método **.emit()** y para recibir el método **.on()**

```
var express = require("express");
var app = express();
var server = require("http").Server(app); //Servidor
var io = require("socket.io")(server); //SocketIO
var siofu = require("socketio-file-upload"); //SocketIO File Uploader

// Construye el HTML del cliente
app.use(express.static(__dirname + "/public"));
app.use(siofu.router);

app.get('/', function(req, res) {
  res.sendFile(__dirname + "/public/login.html");
});

app.get('/publico', function(req, res) {
  res.sendFile(__dirname + "/public/publico.html");
});

app.get('/privado', function(req, res) {
  res.sendFile(__dirname + "/public/privado.html");
});

server.listen(process.env.PORT || 3003, () => {
 console.log("Servidor iniciado en localhost:3003");
});

// Escucha las peticiones de los clientes conectados
io.on("connection", function(socket) {
 /*Aqui va toda la funcionalidad: usuarios, msjes de texto,
 archivos, chat privado, etc... Analogo al for ( ; ; ) en Java*/
});
```

2.2.1. Manejo de usuarios

Cada vez que se conecte un socket cliente, este viene con los siguientes parámetros encapsulados (JSON):

- ID del socket
- Nickname del usuario
- Bandera que indica si su chat es privado o público

Al desencapsular esta información, se guarda el ID del socket y el nickname del usuario en dos arreglos globales (los IDs de sockets son públicos) para que el servidor pueda tener acceso

y control de éstos, respectivamente. La bandera de privado se mantiene en cero (cuando un usuario se conecta por primera vez, entra al chat global).

Luego, el servidor avisa a todos los demás usuarios conectados que un nuevo usuario se ha conectado y actualiza la lista de usuarios en linea, por medio de broadcast (envía la notificación a todos los usuarios excepto al que hizo la solicitud).

```
// Se conecta un cliente
socket.on("user connected", function(nickname) {
 //Revisa que el nickname no este en uso actualmente
 if(nickname in usersOnline)
  console.log("Usuario ya registrado");
 else {
  socket.nickname = nickname;
  socket.privado = 0;
  //Guarda en arreglos
  socketIDPublico[socket.nickname] = socket.id;
  usersOnline.push(socket.nickname);

  console.log(socket.nickname + " conectado. ID: " + socket.id + ". Privado
   ↪  = " + socket.privado);
  //Avisa a los demás clientes que uno nuevo se conecto
  socket.broadcast.emit("user connected", socket.nickname);
  //Actualiza a todos los clientes la lista de usuarios en linea
  sendAllUsersOnline(io, socket);
 }
});
```

2.2.2. Mensajes de texto públicos

Bloque simple. El servidor recibe un mensaje de texto enviado por un usuario (con su respectivo ID de socket), y lo envía por broadcast a los demás de manera pública indicando el remitente (nickname de este usuario). Los emojis vienen como texto.

```
// El cliente envia un mensaje
socket.on("chat message", function(msg) {
 console.log("Msj recibido de: " + socket.nickname + ". Enviando...");

 //El servidor envia el mensaje a todos los demas
 socket.broadcast.emit("chat message", {
  nickname: socket.nickname,
  msg: msg
 });
});
```

2.2.3. Archivos

Para el manejo de archivos, la idea es que cada cliente suba al servidor el archivo que desee compartir (imagenes, audio, pdf, etc), que se almacenará en un directorio específico de éste, y que únicamente se envíe a los demás usuarios el link para su descarga.

Se utiliza el módulo **Socket.IO File Upload** de SocketIO. Primero se crea un objeto que será el manejador para subir los archivos, se le indica en qué ruta guardará éstos y comienza a escuchar un EventListener que se disparará cuando el cliente seleccione un archivo desde su computadora.

Una vez se active tal listener, en el servidor se disparará un evento que indicará si la subida fue exitosa o si existio un error.

```
1  var uploader = new siofu();
2  //El directorio en donde se subiran y encontraran todos los archivos
3  uploader.dir = __dirname + "/public/uploads";
4  uploader.listen(socket);
5
6  // El archivo subido se guardo en el servidor
7  uploader.on("saved", function(e){
8   console.log(e.file);
9  });
10
11 // Ocurrio un error al guardar archivos
12 uploader.on("error", function(e){
13  console.log("Error: " + e);
14 });
15
16 //El cliente envia un link de descarga
17 socket.on("chat file", function(linkMsj) {
18  console.log("Link recibido. Enviando...");
19
20  //El servidor envia el link a todos los demas
21  socket.broadcast.emit("chat file", linkMsj);
22 });
```

Posterior a esto, el servidor envía por broadcast el link del archivo, enviado por el usuario que subió el archivo, a los demás usuarios en línea, que podrán descargarlo (o visualizar si se trata de una imagen).

Toda la documentación del módulo se encuentra en su página web: `https://www.npmjs.com/package/socketio-file-upload`

Este uploader se utilizará tanto para archivos públicos como para archivos de salas privadas.

2.2.4. Salas de chat privadas

La idea del chat privado es que aquel que quiera iniciar una conversación privada con otro usuario, le indique al servidor esto mismo, diciéndole quién será el destinatario.

De este modo, el servidor le puede notificar en cualquier momento al destinatario que un usuario (el remitente) quiere iniciar un chat privado con él, uniéndose ambos a la sala privada.

Así entonces, primero necesitamos una especie de "handshake" del remitente que quiere iniciar un chat privado con el servidor, indicándole quién es (nickname e ID del socket) y con quien quiere hablar (destinatario) para avisarle. La bandera de privado de los sockets cambia a uno (chat privado).

Luego, una vez avisado al destinatario, se crean nuevos IDs de socket para ambos usuarios y se registran en el arreglo de IDs de socket privados. Con esto, no intervenimos en sus IDs públicos (pudiendo un usuario tener chats privados y públicos a la vez).

Registrados ambos usuarios/sockets en la "sala privada", el envío de mensajes de texto, emojis, y archivos de la misma manera, pero teniendo en cuenta que ahora el contenido es recibido por un remitente y se enviará únicamente al socket con el ID del destinatario (indicando el remitente), y no a broadcast.

```
// Un cliente quiere iniciar un chat privado con otro - HandShake
socket.on("start private", function(room) {
 console.log(room.from + " quiere iniciar un chat privado con " + room.to);

 // Enviamos notificacion unicamente al destinatario
 socket.to(socketIDPublico[room.to]).emit("start private chat", {
  from: room.from,
  to: room.to,
 });
});

// Se registran los nuevos IDs de socket privados
socket.on("private IDs", function(nickname) {
 socket.nickname = nickname;
 socket.privado = 1;
 //Usuarios se unen al chat privado con un ID de socket nuevo - privado
 console.log(socket.nickname + " unido a chat privado con nueva ID: " +
  ↪  socket.id + ". Privado = " + socket.privado);
 //Se registra el ID en el arreglo de privados
 socketIDPrivado[socket.nickname] = socket.id;
});

// El cliente envia un mensaje privado
socket.on("private chat message", function(data) {
```

```
24  console.log("Msj privado recibido de: " + socket.nickname + ". Enviando a "
     ↪  + data.to);
25
26  //El servidor busca el ID privado del destinatario por medio de su nickname
27  var id = socketIDPrivado[data.to];
28  //El servidor envia el mensaje, junto al remitente, unicamente al
     ↪  destinario
29  socket.to(id).emit("private chat message", {
30   nickname: socket.nickname,
31   msg: data.msg
32  });
33 });
34
35 // El cliente envia un link de descarga privado
36 socket.on("private chat file", function(data){
37  console.log("Link privado recibido. Enviando...");
38  //El servidor busca el ID privado del destinatario por medio de su nickname
39  var id = socketIDPrivado[data.to];
40  //El servidor envia el link unicamente al destinario
41  socket.to(id).emit("private chat file", data.link);
42 });
```

2.2.5. Desconexión de usuarios

Según la documentación de SocketIO, cada vez que se abre una nueva ventana en el navegador del cliente, o se actualiza la actual, el framework crea un nuevo socket.

En consecuencia, cada vez que un cliente usuario cierra o se desconecta del chat, el socket perteneciente a este se destruye, así como su ID.

Para tener un mejor control de éste evento, que por desgracia no podemos configurar directamente. hacemos uso de la bandera privado.

Si la bandera vale 1 y se dispara éste evento en el servidor, significa que un **un usuario abandono únicamente una sala de chat privado**, por lo que únicamente notificamos de esto a los miembros de la sala privada restantes que aún estén dentro de ella.

Si la bandera vale 0 y se dispara éste evento en el servidor, significa que un **un usuario abandono el chat público, o abandono todas sus salas privadas y el chat público**. Aquí si notificamos a todos los usuarios en línea en el chat público, removemos su nickname de la lista de usuarios en línea, y enviamos la actualizada nuevamente al resto de usuarios.

```
1  //El cliente se desconecta
2  socket.on("disconnect", function() {
3   if(socket.privado == 1) {
4    //El usuario abandono una sala privada, no se avisa en el chat publico
```

```
    socket.privado = 0;
    console.log(socket.nickname + " abandono una sala privada. Privado = " +
     ↪ socket.privado);

    socket.broadcast.emit("user leaves", socket.nickname);
    socket.broadcast.emit("enable user", socket.nickname);
   }
   else {
    //El usuario se fue de la aplicacion, se avisa en el chat público
    console.log(socket.nickname + " desconectado. Privado = " +
     ↪ socket.privado);

    usersOnline.splice(usersOnline.indexOf(socket.nickname), 1);
    //El servidor notifica todos los demas
    socket.broadcast.emit("user disconnected", socket.nickname);
    sendAllUsersOnline(socket);
   }
});
```

2.3. Configuración del Cliente

Del lado del cliente, declaramos una variable socket, por medio de la cuál estaremos enviando y recibiendo información del servidor, y creamos un objeto uploader para subir archivos a este. También indicamos el host y el puerto al cuál nos conectaremos.

```
$(function() {
  //var rootPath = "https://chat-multicast.herokuapp.com/";
  var rootPath = "http://localhost:3003/";
  var socket = io();
  // ---------------------- SUBIR ARCHIVOS
  var uploader = new SocketIOFileUpload(socket);
}
```

2.3.1. Manejo de usuarios

Cuando un cliente se conecta al servidor, éste le pide un nickname para como identificación de usuario. Una vez ingresado el nickname, el socket cliente lo envía al servidor para su registro y le permite el acceso al chat público.

El socket recibe los usuarios que están en línea desde el servidor y los despliega.

Además, cada vez que un nuevo usuario se conecte desde otro navegador, computadora, pestaña, etc. el socket cliente recibe del servidor la lista actualizada de los usuarios en línea con el nuevo que se recién se unió.

Por último, cada vez que un usuario se desconecte, el socket cliente recibirá el aviso desde el servidor, con la lista actualizada de usuarios, y desplegará esta información en la interfaz.

El cliente irá contabilizando el número de usuarios en línea y los desplegará cada vez que ocurran estos eventos.

```
var numUsers = 0; //Contabilizar usuarios en línea

//Envia al servidor el nickname del usuario que se acaba de conectar al chat
 ↪ publico
socket.emit("user connected", nickname);
console.log("Enviando nickname al servidor");

//Recibe del servidor, muestra y va actualizando la lista de usuarios online
socket.on("display users", function(users) {
 console.log("Recibiendo usuarios: " + users);
 $("#users li").remove();
 numUsers = 0;
 //Contabiliza a los demás usuarios en línea, y actualiza la interfaz HTML
  ↪ mostrando la lista de usuarios online
 for(var i = 0; i < users.length; i++){
  if(users[i].localeCompare(nickname) != 0) {
      numUsers++;
      // Va añadiendo los usuarios
      $("#users").append($('<li id="'+ users[i] +'" class="users-li"><i
       ↪ class="fas fa-chevron-circle-right"
       ↪ style="font-size:26px;color:#52de7a"></i><button class="btn-users"
       ↪ id="' + users[i] + '">' + users[i] + '</button></li>'));
   }
 }

 //Muestra el número de usuarios en línea en la interfaz HTML
 if(numUsers > 0)
  $("#messages").append($("<li style=background:#52de7a; >").text(numUsers +
   ↪ " usuarios en línea "));
 else
  $("#messages").append($("<li>").text("No hay usuarios en linea " +
   ↪ String.fromCodePoint(emojis[40])));

 ajustarScroll(); //Ajusta la vista para visualizar los mensajes más
  ↪ recientes
});

//Agrega a los nuevos usuarios cuando se conecten, notificado por el
 ↪ servidor
```

```
31 socket.on("user connected", function(newNickname) {
32  console.log("Usuario: " + newNickname + " conectado");
33
34  $("#messages").append($("<li style=background:#52de7a; >").text(newNickname
     ↪  + " se ha conectado"));
35  ajustarScroll();
36 });
37
38 //Elimina y envia al servidor el usuario desconectado
39 socket.on("user disconnected", function(nickname) {
40  console.log("Usuario: " + nickname + " desconectado");
41
42  //Actualiza la lista de usuarios online en el HTML, quitando el nickname
     ↪  del usuario
43  $("#messages").append($("<li style=background:#52de7a;>").text(nickname + "
     ↪  se ha desconectado " + String.fromCodePoint(emojis[40])));
44  $("#" + nickname).remove();
45
46  //Contabiliza nuevamente la cantidad de usuarios y la muestra
47  numUsers--;
48  if(numUsers > 0)
49   $("#messages").append($("<li>").text(numUsers + " usuarios en línea"));
50  else
51   $("#messages").append($("<li>").text("No hay usuarios en linea" +
      ↪  String.fromCodePoint(emojis[40])));
52
53  ajustarScroll();
54 });
```

2.3.2. Mensajes de texto públicos

Este bloque también es muy simple. El usuario escribe el texto por medio de la interfaz HTML, y ya sea presionando el botón de "Enviar" o dando Enter, el cliente envía el mensaje al servidor, y como se explico anteriormente, este reenvía por broadcast el mensaje a todos los demás, indicando el remitente.

El cliente agrega el mensaje del remitente a su vista por cuestiones de estética, pero no al servidor.

Además, el cliente esta a la espera de nuevos mensajes que sean enviados por los demás, siendo recibidos desde el servidor.

```
1 //Envia mensajes dando clic al boton
2 $("#enviarMensaje").click(function(e) {
3   //Obtiene el contenido del msj de la vista
```

```
4   var msj = $("#m").val();
5
6   if(msj != ""){
7     //Envia el mensaje al servidor
8     socket.emit("chat message", msj);
9     //Agrega el mensaje enviado a la vista, pero no se lo reeenvia a
      ↪ sí mismo
10    $("#messages").append($("<li>").text(nickname + ": " +msj));
11    $("#m").val("");
12    //Ajusta la vista para ver los mensajes más recientes
13    ajustarScroll();
14  }
15 });
16
17 //Envia mensajes al presionar Enter. Misma lógica....
18 $(window).keydown(function(e) {
19   if (e.which === 13) {
20     var msj = $("#m").val();
21
22     if(msj != ""){
23       socket.emit("chat message", msj);
24       $("#messages").append($("<li>").text(nickname + ": " +msj));
25       $("#m").val("");
26       ajustarScroll();
27     }
28   }
29 });
30
31 // Recibe del servidor los mensajes de otros usuarios
32 socket.on("chat message", function(data) {
33   console.log("Msj: " + data.msg + " de: " + data.nickname);
34   //Agrega los mensajes recibidos a la interfaz
35   $("#messages").append($("<li>").text(data.nickname + ": " + data.msg));
36   ajustarScroll();
37 });
```

2.3.3. Emojis

Los emojis son simples códigos hexadecimales que se envían como texto tal cual como en el bloque anterior. Solo debemos de habilitarle el formato UTF-8 a nuestro HTML y listo.

Se agregan todos los emojis a la vista, cada uno con un EventListener, de modo que si se da clic en ellos, se agregan al mensaje de texto a enviar. También se pueden agregar haciendo copy-paste directamente.

La lista completa de emojis en código hexadecimal es la siguiente:

```
let emojis = ['0x1F600', '0x1F603', '0x1F604', '0x1F601', '0x1F606',
              '0x1F605', '0x1F923', '0x1F602','0x1F642', '0x1F643',
              '0x1F609', '0x1F60A', '0x1F607', '0x1F60D', '0x1F929',
              '0x1F618', '0x1F617', '0x1F61A', '0x1F619', '0x1F60B',
              '0x1F61B', '0x1F61C', '0x1F92A', '0x1F610', '0x1F61D',
              '0x1F911', '0x1F917', '0x1F92D', '0x1F92B', '0x1F914',
              '0x1F910', '0x1F928', '0x1F611', '0x1F636', '0x1F60F',
              '0x1F612', '0x1F644', '0x1F62C', '0x1F925', '0x1F60C',
              '0x1F614', '0x1F62A', '0x1F924', '0x1F634', '0x1F637',
              '0x1F912', '0x1F915', '0x1F922', '0x1F92E', '0x1F927',
              '0x1F635', '0x1F92F', '0x1F920', '0x1F60E', '0x1F913',
              '0x1F9D0', '0x2639', '0x1F62E', '0x1F632', '0x1F633',
              '0x1F626', '0x1F627', '0x1F628', '0x1F630', '0x1F625',
              '0x1F622', '0x1F62D', '0x1F631', '0x1F616', '0x1F623',
              '0x1F61E', '0x1F613', '0x1F629', '0x1F62B', '0x1F624',
              '0x1F621', '0x1F620', '0x1F92C', '0x1F608', '0x1F47F',
              '0x1F480', '0x2620', '0x1F4A9', '0x1F47D', '0x1F648',
              '0x1F649', '0x1F64A', '0x1F496', '0x1F497', '0x1F493',
              '0x1F49E', '0x1F495', '0x1F494', '0x2764', '0x1F49B',
              '0x1F49A', '0x1F499', '0x1F4AF', '0x1F44C'];

//Cargar los emojis en la vista
function cargarEmojis() {
  for(let i = 0; i < emojis.length; i++) {
    $("#emojis").append($('<li class ="emojis" id="'+ i +
     ↪  '"onclick="ponerEmoji(this)">').text(String.fromCodePoint(emojis[i])));
    $("#emojis").append($("</li>"));
  }
}

//Para agregar los emojis en el mensaje a enviar
function ponerEmoji(element) {
  var msj = $("#m").val();
  var i = $(element).attr("id");
  // Concatena el mensaje actual mas el emoji
  var mensaje = msj + String.fromCodePoint(emojis[i]);
  $("#m").val(mensaje);
}
```

2.3.4. Archivos

Antes que nada, debemos de agregar un EventListener a un botón, de tal forma que cada vez que se de clic en él, se abra una ventana para seleccionar el archivo a enviar.

Posteriormente, al uploader que previamente declaramos en la configuración del cliente le agregamos dos EventListeners:

1. *Progress:* Para indicarle al usuario el porcentaje de subida de su archivo
2. *Succed:* Cuando el archivo se ha subido satisfactoriamente al servidor. Dentro de éste, también construiremos el link de descarga para los demás usuarios, y lo enviaremos al servidor para que éste lo reenvíe por broadcast.

Si el archivo a subir se trata de una imagen, ésta se mostrará en la interfaz (además de su link de descarga).

El navegador web se encargará de la descarga de archivos almacenados en el servidor.

```
1  //EvenListener para automaticamente cargar el archivo de la computadora del
    ↪  usuario desde un boton
2  document.getElementById('subir').addEventListener("click", uploader.prompt,
    ↪  false);
3
4  //Muestra el progreso de subida en la interfaz, recibe el EventListener como
    ↪  parametro
5  uploader.addEventListener("progress", function(e) {
6    //Datos del archivo obtenidos desde el EventListener
7    var nombre = e.file.name;
8    var tam = e.file.size;
9    var porcentaje = parseInt(e.bytesLoaded / e.file.size * 100);
10   console.log("Nombre: " + nombre);
11   console.log("Tam: " + tam);
12   console.log("Subiendo: " + porcentaje + "%");
13
14   $("#messages").append($("<li>").text("Enviando " + nombre + " de " + tam +
      ↪  " bytes: " + porcentaje + "%"));
15   ajustarScroll();
16 });
17
18 //Envia link de descarga a otros clientes una vez finalizada la subida
19 uploader.addEventListener("complete", function(e) {
20   console.log(e.success);
21
22   if(e.succes != 1) {
```

```
23      var nombre = e.file.name;
24      var tam = e.file.size;
25      console.log(nombre);
26      console.log(nombre.length);
27
28      let aux = "";
29      for(let i = 0; i < nombre.length; i++) {
30        let charNombre = nombre.charAt(i);
31        if(charNombre == ".") {
32          for(let j = i; j < nombre.length; j++) {
33            charNombre = nombre.charAt(j);
34            aux = aux + charNombre;
35          }
36          break;
37        }
38      }
39      console.log("extension: " + aux);
40
41      //Si se trata de una imágen, envia un <img></img> junto al link de
         ↪ descarga
42      if(aux === ".PNG" || aux == ".jpg" || aux == ".JPG" || aux == ".png" ||
         ↪ aux == ".jpeg" || aux == ".JPEG") {
43       var ancho = $(window).width() * 0.2;
44        var imagenFile = '<li>' + nickname + ': ' + '<img class =
           ↪ "imagen-chat" src="' + rootPath + 'uploads/' + nombre + '"
           ↪ width="' + ancho + '"></li>';
45        console.log("Envio mi imagen a MOSTRAR CHAT");
46
47        //Envia
48        socket.emit("chat file", imagenFile);
49        $("#messages").append($(imagenFile));
50      }
51      else{
52        //Si no es imagen, solo se envia el link de descarga
53       var msjFile = '<li>' + nickname + ': ' + '<a href="' + rootPath +
           ↪ 'uploads/' + nombre + '" download>' + nombre + '. ' + tam + '
           ↪ bytes</a></li>';
54       //Envia
55        socket.emit("chat file", msjFile);
56       $("#messages").append($(msjFile));
57      }
58      ajustarScroll();
59    }
60  });
```

2.3.5. Salas de chat privadas

Recordemos que en la vista HTML, constantemente se esta desplegando y actualizando la lista de usuarios en línea actualmente. La idea de chats privados es que, cuando el usuario quiera iniciar una conversación privada con otro usuario, éste de clic en su nickname, dentro de la lista de usuarios, y que el cliente avise al servidor que se quiere iniciar una sala privada (handshake).

Posteriormente, el servidor notifica al usuario destinatario, y ambos usuarios se registran en una sala privada en el servidor.

Una vez ya unidos ambos usuarios a la sala privada, el servidor notifica esto a ambos clientes, y éste último abre otra nueva pestaña del navegador de los dos clientes, al mismo tiempo, donde podrán empezar la conversación privada.

Adicionalmente, se tendrá un arreglo que contendrá el nickname de aquellos usuarios que actualmente estén en una sala privada, para así evitar salas repetidas.

Cada vez que un usuario ingrese a una sala privada, si el remitente quiere abrir otra nueva sala pero con el mismo destinatario, ésta acción será ignorada, hasta que se cierre la ventana de dicha sala. Será hasta que se cierre la ventana cuando éste destinatario estará nuevamente disponible en la lista de usuarios en línea para una nueva conversación privada.

Finalmente, cuando un usuario abandona una sala privada, el servidor únicamente notifica en ésta sala.

Toda la funcionalidad de envío de mensajes, usuarios, emojis, archivos es la misma que la del chat público (sólo que el servidor reenvía todo al canal privado, y no al público).

```
1  //Abrir nueva pestaña en el navegador
2  function abrirSalaPrivada(rootPath, from, to) {
3    //Se revisa si el usuario destinatario ya se encuentra en un chat privado
       ↪ con el usuario (remitente)
4    if(jQuery.inArray(to, usersPrivado) == -1){
5      //Se agrega el nickname del usuario en el arreglo de usuarios en chat
         ↪ privado actualmente
6      usersPrivado.push(to);
7
8      //Se abre la nueva pestaña, por medio de un pop-up
9      var newRoom = window.open(rootPath + "privado/?private=1&from=" + from
         ↪ +"&to=" + to, "_blank");
10     if (!newRoom)
11       alert('Por favor active los pop-ups del navegador');
12   }
13   else
14     console.log("Chat privado con: " + to + " ya existe.")
15 }
16
17 //Iniciar chat privado con algun usuario al dar clic en él - HandShake
```

```
18 $(document).on("click", ".btn-users", function() {
19   var to = $(this).attr("id");
20   console.log("Enviando solicitud de chat privado a: " + to);
21
22   var participantes = {
23     from: nickname,
24     to: to
25   };
26
27   //Envia al servidor los participantes
28   socket.emit("start private", participantes);
29   //Por parte del REMITENTE, abre automaticamente la nueva ventana de chat
     ↪ privado
30   abrirSalaPrivada(rootPath, nickname, to);
31 });
32
33 //Se recibe una solicitud para iniciar un chat privado
34 socket.on("start private chat", function(room) {
35   console.log("Recibida solicitud para unirse al chat privado de: " +
     ↪ room.from);
36   //Por parte del DESTINATARIO
37   abrirSalaPrivada(rootPath, room.to, room.from);
38 });
39
40 //El servidor indica que usuario ha abandonado el chat privado, por lo que
   ↪ es habilitado de la lista de usuarios en línea
41 socket.on("enable user", function(user){
42   console.log("Usuario: " + user + " habilitado para privado.");
43   //Se elimina el nickname del usuario del arreglo
44   usersPrivado.splice(usersPrivado.indexOf(user), 1);
45 })
```

3. Pruebas

La aplicación web esta disponible para todo público, ingresando al siguiente link: `https://chat-multicast.herokuapp.com/`

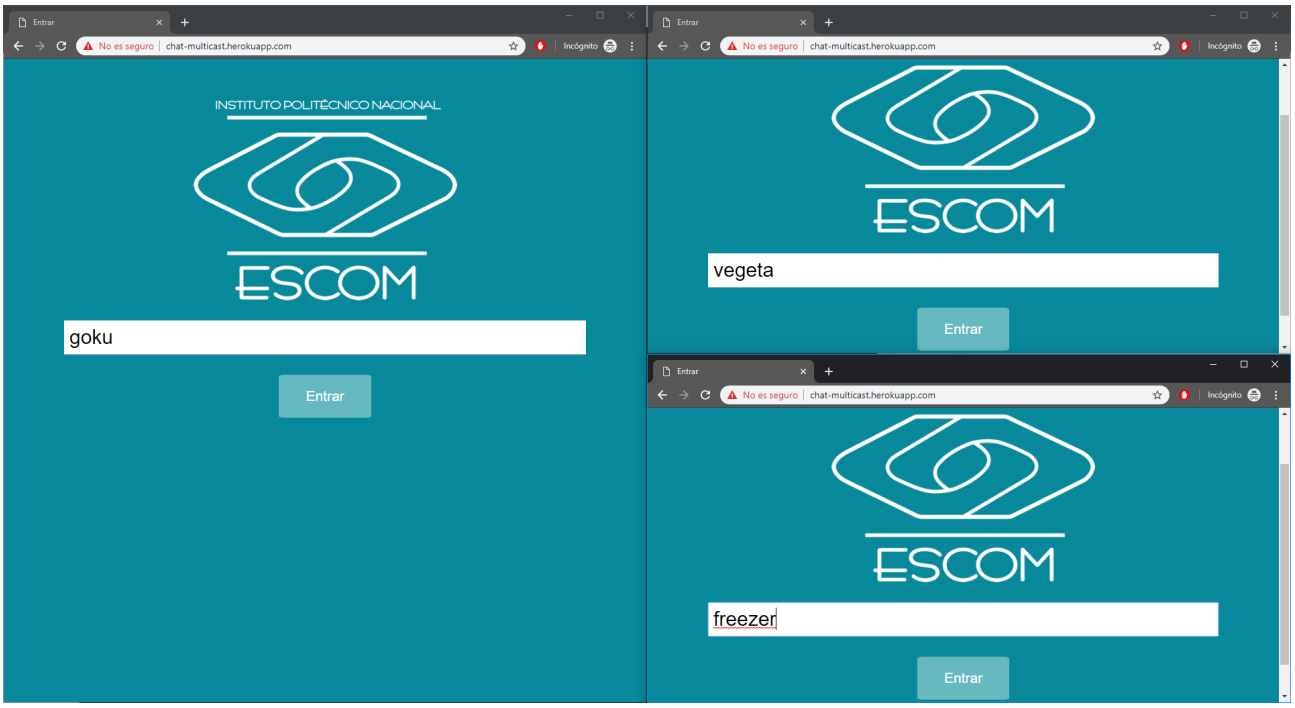

Figura 1: Inicio

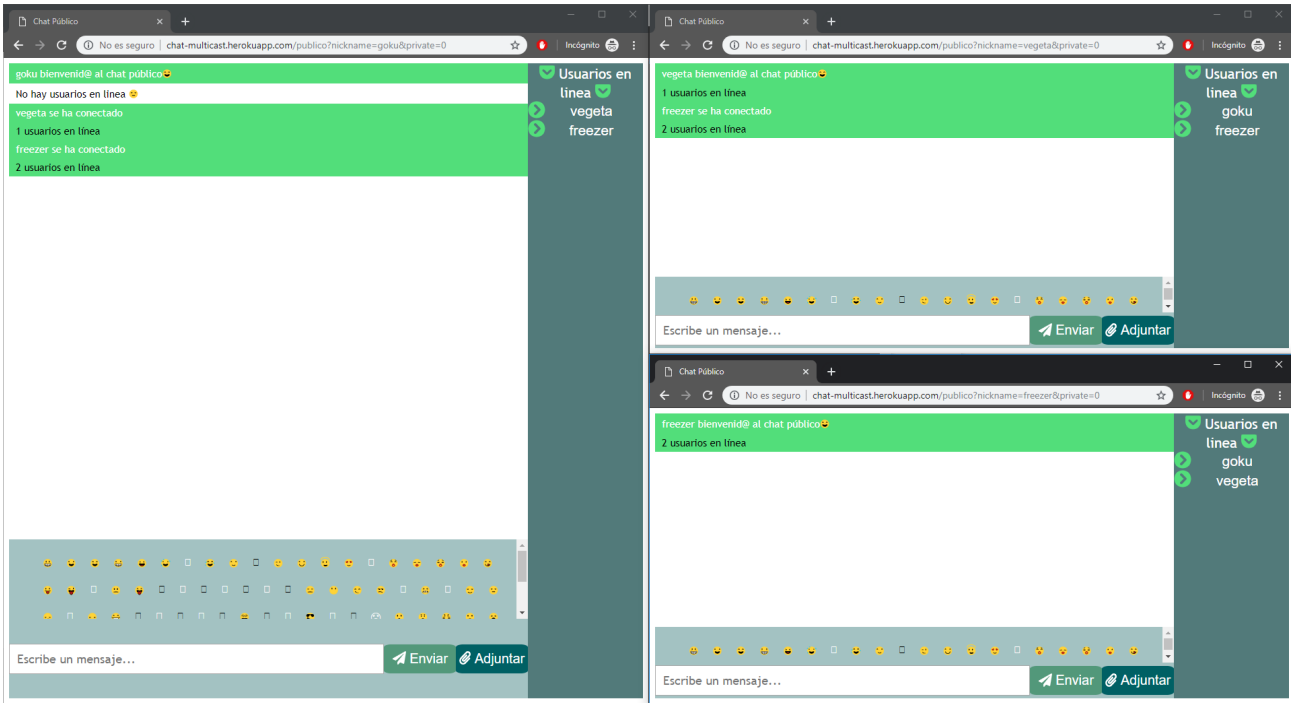

Figura 2: Chat grupal

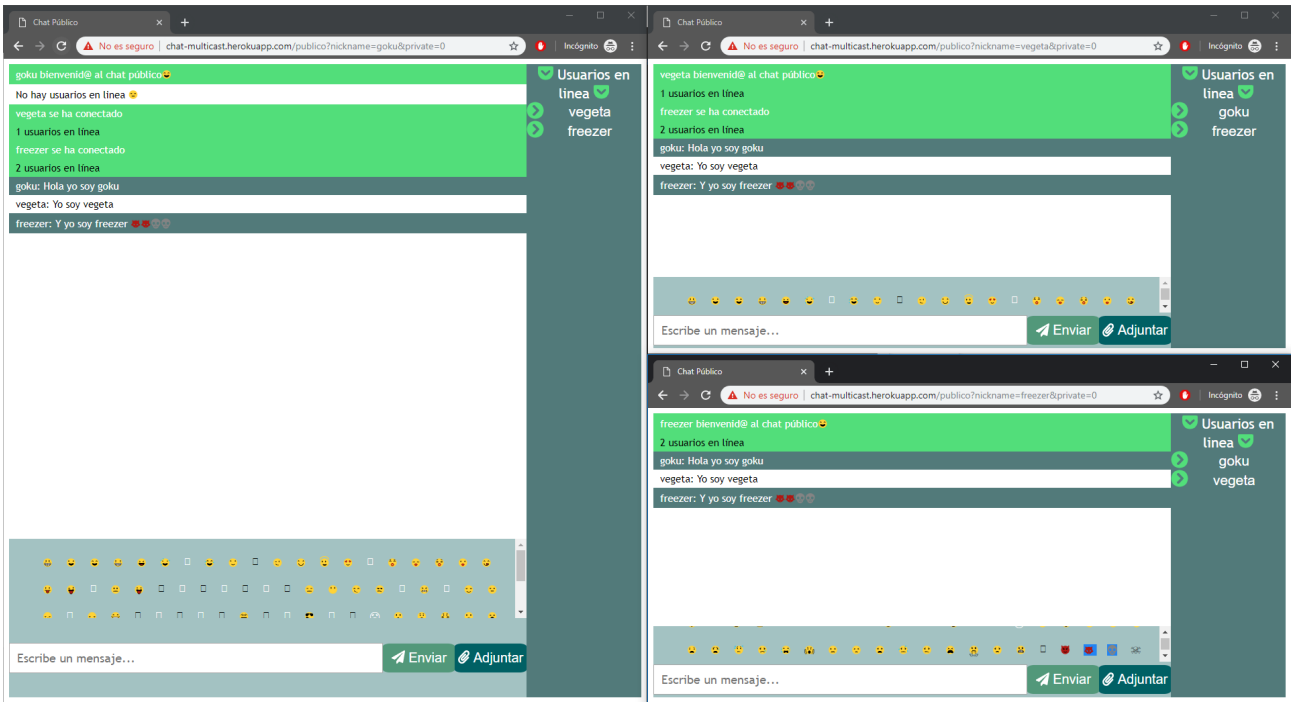

Figura 3: Usuarios conectados y lista de usuarios en línea

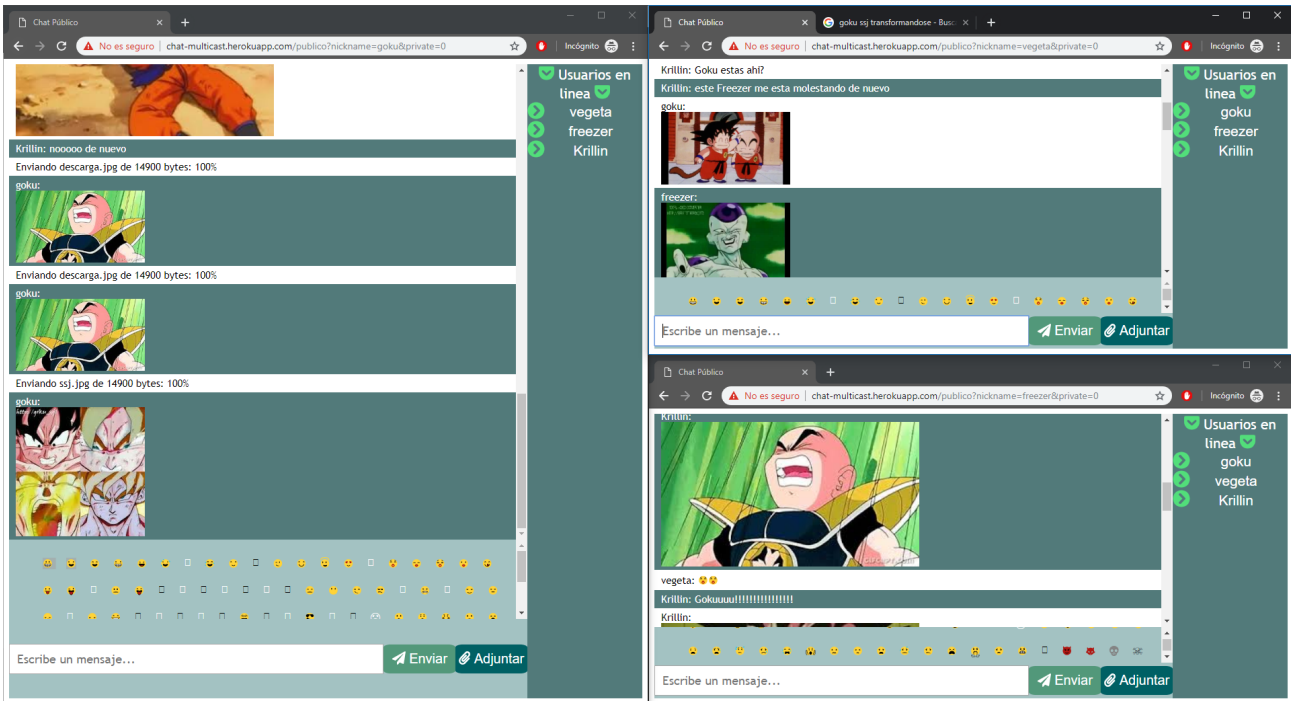

Figura 4: Envío de imágenes

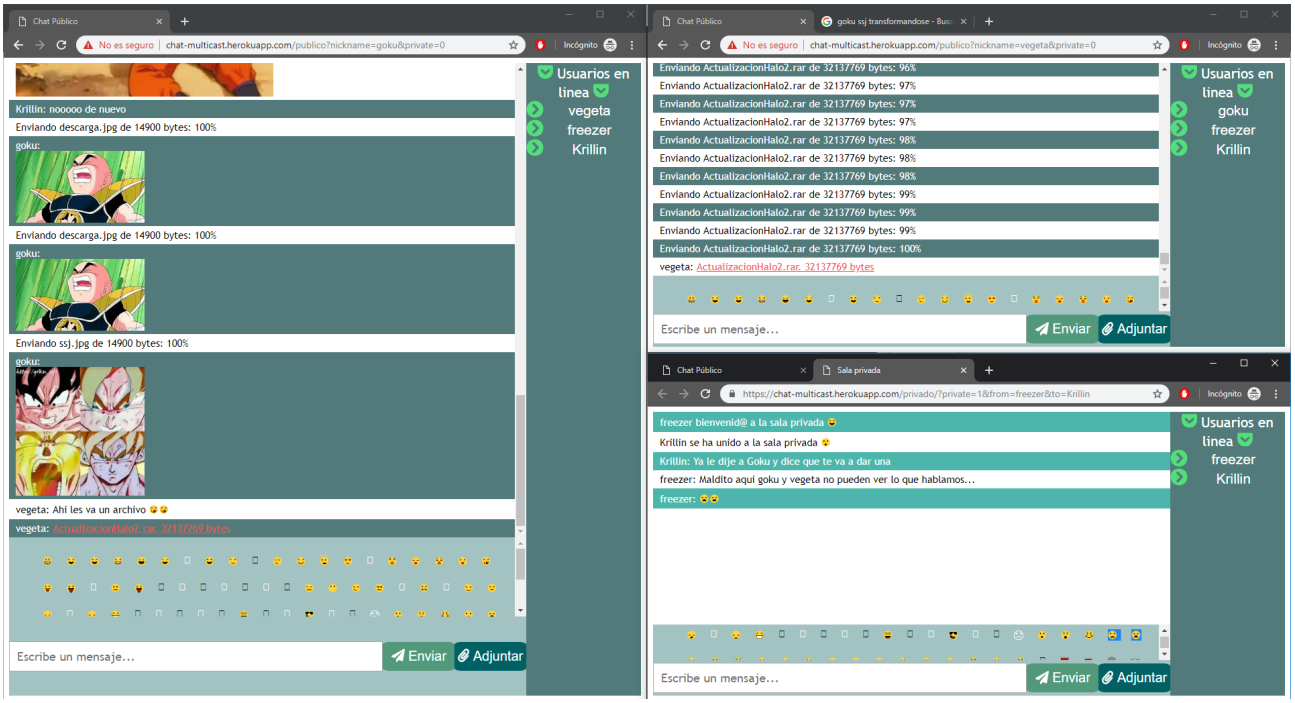

Figura 5: Envío de archivos con su link de descarga

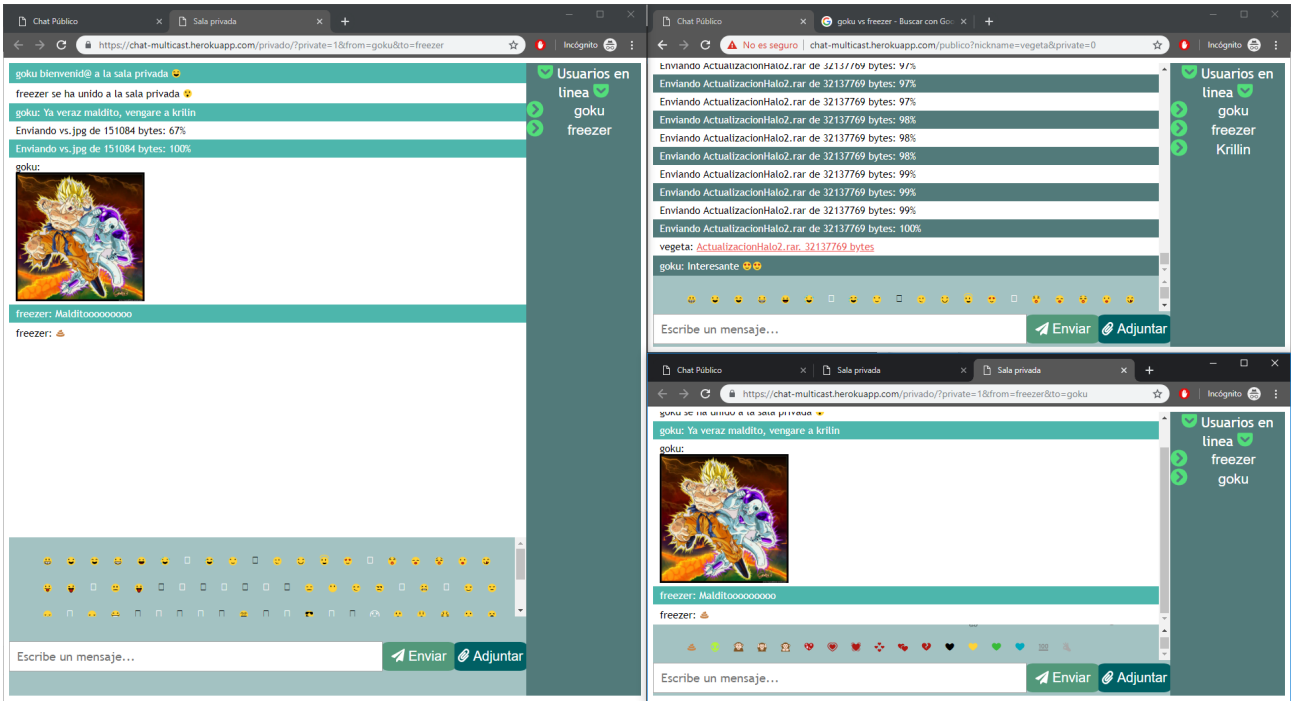

Figura 6: Sala de chats privados

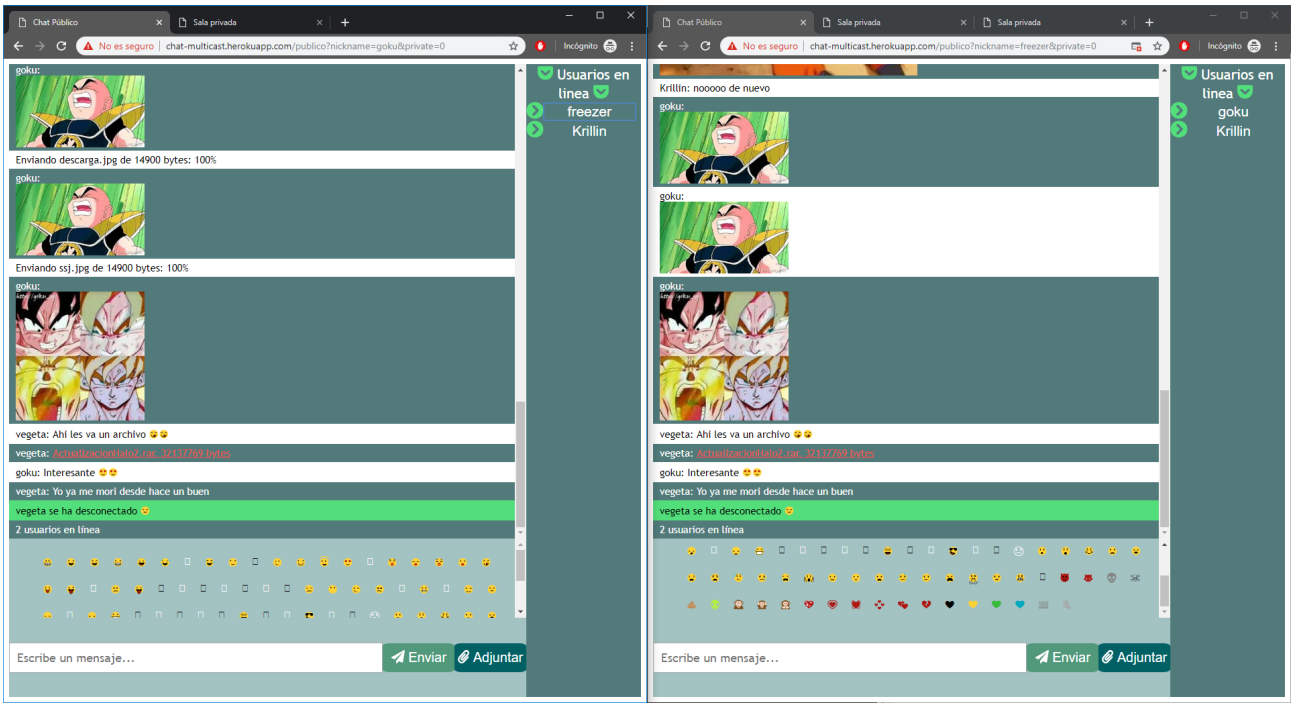

Figura 7: Envío de imágenes en chat público

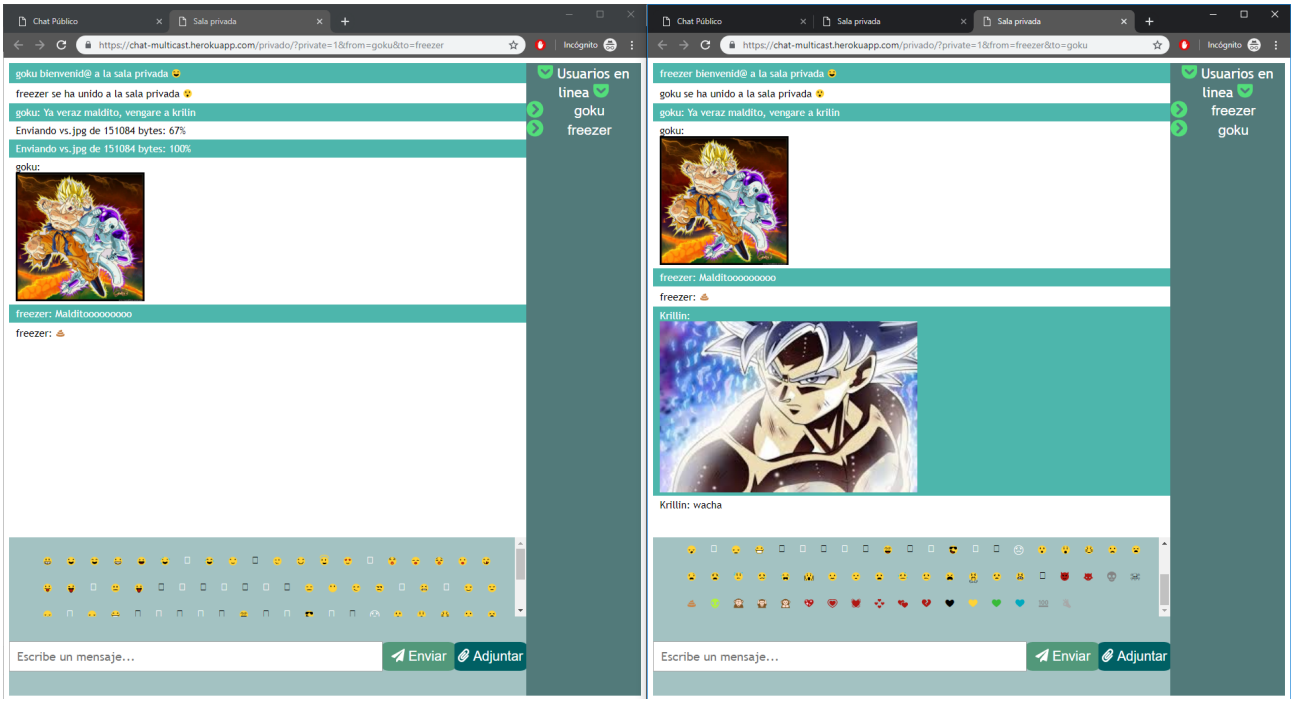

Figura 8: Envío de imágenes en chat privado

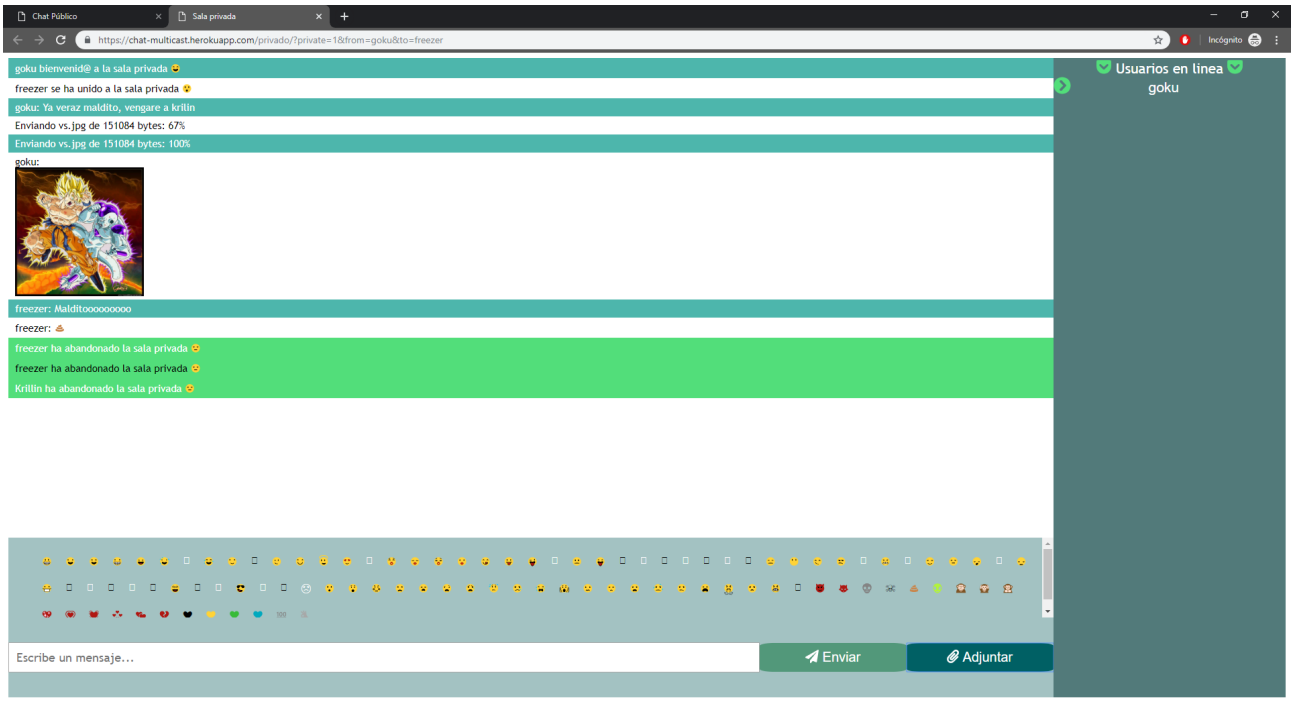

Figura 9: Desconexión de usuarios

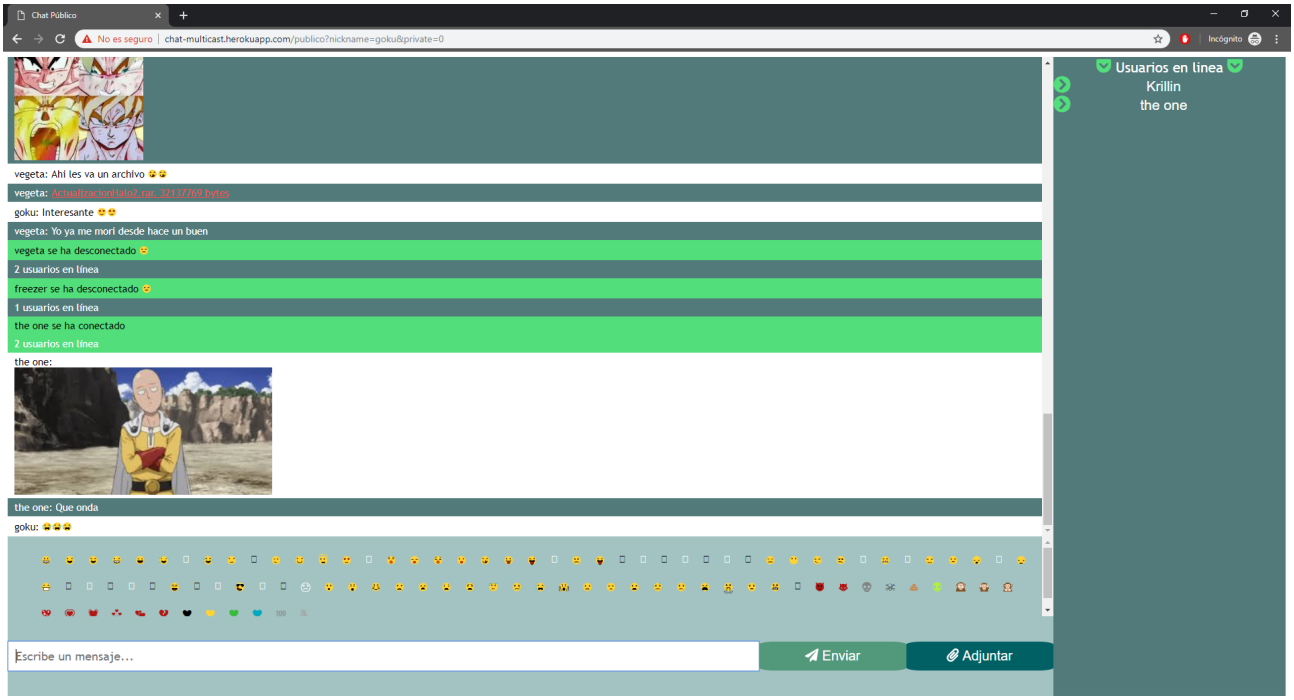

Figura 10: Nuevos usuarios conectados

4. Posibles mejoras

1. Añadir una base de datos para guardar los mensajes antiguos, y desplegarlos cada vez que se accedan a los chats, públicos y privados.
2. Dar soporte de vistas previas para videos, gifs, audios, similar a las imágenes.
3. Añadir mayor personalización para los usuarios: cuenta personal, perfil, foto o avatar, colores predefinidos.
4. Tener gestión de usuarios, sesiones, etc.
5. Permitir a los usuarios ver los archivos que han enviado, así como filtrar mensajes.
6. Evitar usuarios con nicknames repetidos.
7. Búsqueda de mensajes y usuarios.
8. Darle responsividad y mejorar el diseño de las vistas HTML.
9. Mejorar el despliegue al usuario del progreso de subida de archivos.
10. Permitir salas de chat privadas con más de dos usuarios.
11. Agregar más emojis y clasificarlos.

5. Conclusiones

5.1. Nicolás Sayago Abigail

El aprendizaje adquirido en esta práctica no fue únicamente sobre sockets, también aprendimos a usar una tecnología WEB que nos ayudará para próximos proyectos. Existen implementaciones que después de haberlas hechas pensamos que trabajamos doble. Aprendimos a poner emojis, crear conversaciones privadas y públicas.

5.2. Ramos Diaz Enrique

En ésta práctica decidimos probar una nueva tecnología distinta a la que veníamos trabajando en prácticas anteriores. Con ayuda de frameworks y bibliotecas, el envío, recibo y reenvío por parte del servidor de mensajes de texto, archivos, emojis, etc. se nos facilitó de forma considerable. Tuvimos ciertas complicaciones en separar los canales públicos y privados, ya que el manejo y creación de los ID de los sockets no funciona de una forma como nosotros la imaginamos en un principio. Para facilidad de uso y comodidad, subimos la aplicación a un servidor, para así poder acceder al chat comunal en cualquier momento, desde cualquier dispositivo que soporte navegadores web.