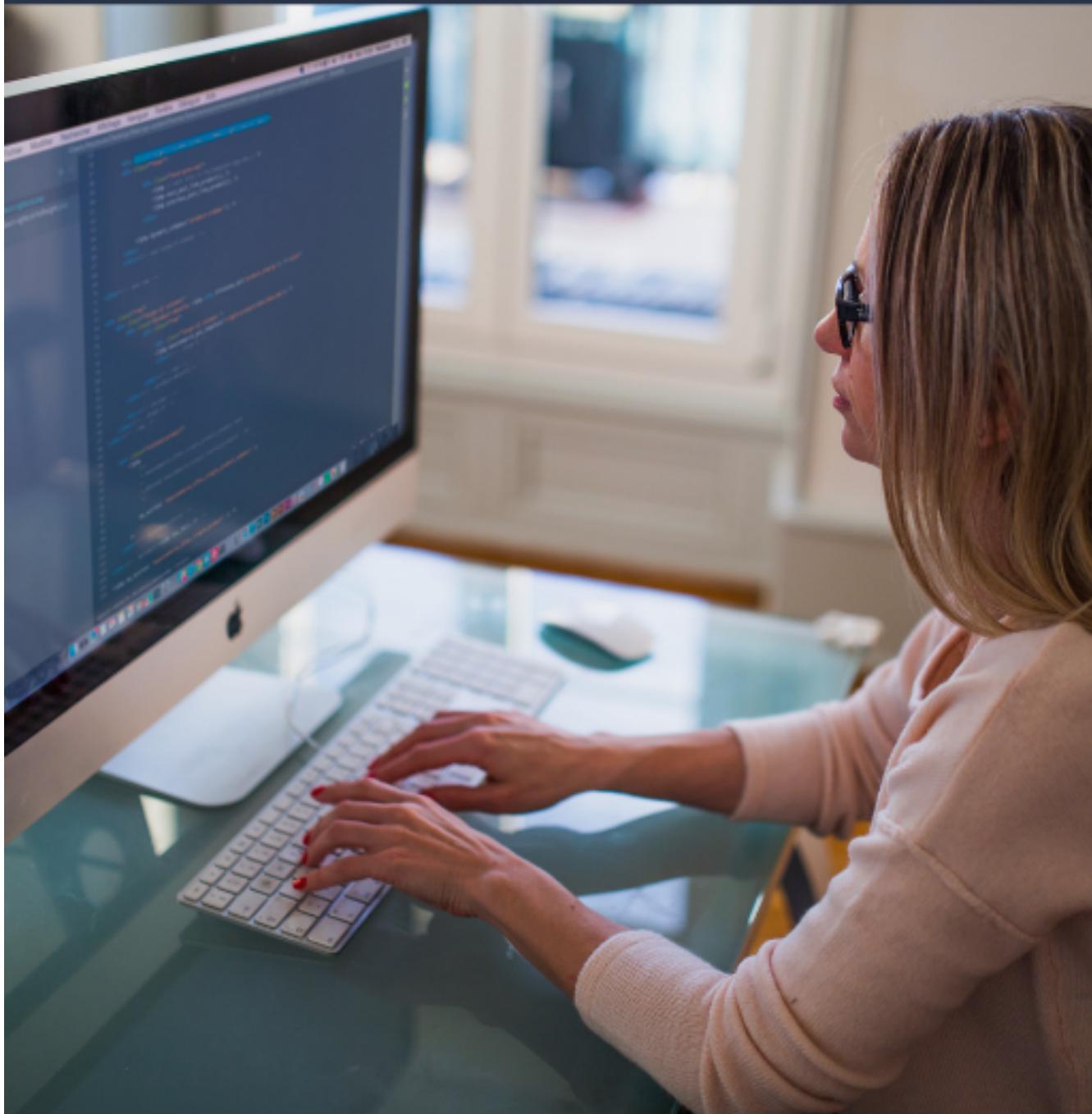


PROGRAMACIÓN DISTRIBUIDA DE SERVICIOS DE INTERNET



UNIVERSIDAD DE COLIMA
FACULTAD DE TELEMATICA

CHAT

ALUMNO: AQUINO RODRÍGUEZ DAFNE
PROFESOR: MONTAÑO ARAUJO SERGIO ADRIAN

ÍNDICE:

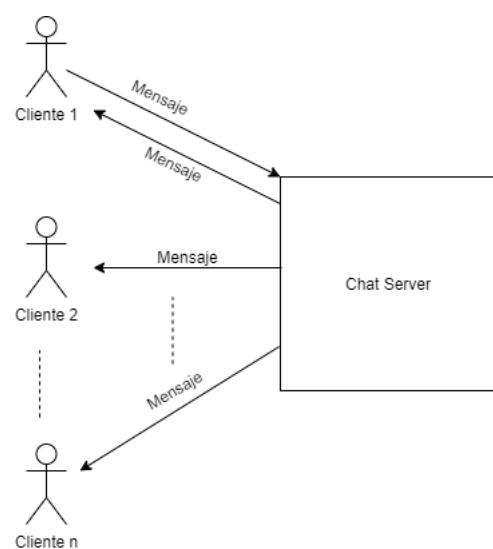
INTRODUCCIÓN.....	2
DESARROLLO	
• Librerías utilizadas.....	3
•	
• index.js.....	4
• index.html.....	7
• chat.sql.....	13
• ¡RESULTADOS!	13
GLOSARIO.....	15
CONCLUSIÓN.....	16

INTRODUCCIÓN:

Actualmente, la tecnología se encuentra en todos lados y la utilizamos día con día, por ejemplo, al hacer uso de alguna red social a través de nuestro celular.

Con socket podemos construir aplicaciones con conexión persistente entre un cliente y un servidor ya que permite la comunicación en tiempo real, bidireccional y basada en evento, funciona en todas las plataformas, navegadores o dispositivos.

En esta actividad veremos un ejemplo de su funcionamiento con ayuda de un chat el cual trabaja de la manera que se muestra en la imagen a continuación.



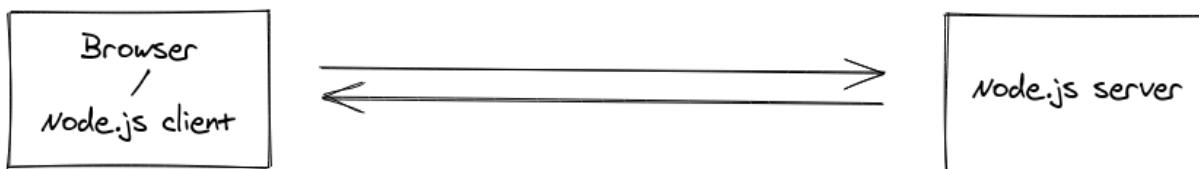
Librerías utilizadas:

- ❖ npm init

Utilizado para configurar un paquete npm nuevo o existente, creará o actualizará el package.json y ejecutará cualquier otra operación relacionada con la inicialización.

- ❖ npm socket.io

Es una librería open source con una amplia comunidad que nos ayudará a construir aplicaciones con conexión persistente entre cliente y servidor en tiempo real.



El cliente intentará establecer una conexión WebSocket si es posible.

WebSocket: Protocolo de comunicación que proporciona un canal full-duplex entre servidor y navegador.

- ❖ npm install mysql

MySQL es una base de datos la cual puede ser accedida desde Node.js, por ende, se instala un controlador.

- ❖ npm install express

Express es un framework para Node.js que sirve para ayudarnos a crear aplicaciones web en menos tiempo, ya que nos proporciona funcionalidades como el enrutamiento y opciones para gestionar sesiones.

- ❖ npm install express-session

Archivos:

✓ index.js

En el archivo .js como primer paso agregamos los paquetes de la aplicación de Node.js y módulos.

```
const express = require('express'), //Agregamos los paquetes de la aplicación de Node.js y modulos
socket = require('socket.io'),
mysql = require('mysql');
cookieParser = require('cookie-parser'), //Se utiliza para manejar las sesiones
session = require('express-session');

var app = express(); //Paquetes para la aplicación del desarrollo web, como sesiones y manejo de solicitudes HTTP
```

Posteriormente levantamos el puerto y mandamos un mensaje que se escucha el puerto en consola, en nuestro navegador escribiremos "localhost:3030", esto indica que 3030 es el puerto con el que estamos trabajando.

```
var server = app.listen(3030, function () { //Indicar que se escucha el puerto, utilizando el puerto 3030
  console.log("Servidor en marcha, port 3030.");
});
```

Como tercer paso le indicamos a Express que utilizaremos algunos de sus paquetes.

```
var sessionMiddleware = session({
  secret: "keyUltraSecret",
  resave: true,
  saveUninitialized: true
});
```

Utilizamos Socket para pasar las sesiones.

```
io.use(function (socket, next) { //Pasamos las sesiones con el socket
  sessionMiddleware(socket.request, socket.request.res, next);
});
```

Pasamos a conectar la base de datos y verificamos si la base de datos existe, su ruta es correcta, entre otros componentes importantes de conexión, de lo contrario mandamos un mensaje de error.

```
const config = {
  host: "localhost",
  user: "root",
  password: 'password',
  base: "chat"
};
```

```
var db = mysql.createConnection({
  host      : 'localhost',
  user      : 'root',
  password  : 'password',
  database : 'chat'
});
```

```
db.connect(function (err) {
  if (!!err)
    throw err; //Me
```

Hacemos una función para crear la conexión del usuario, con un mensaje en consola imprimimos las cookies y busca si existe un user con ese ID.

```
io.on('connection', function (socket) {  
  var req = socket.request;  
  
  console.log(req.session);
```

Al iniciar la sesión hacemos una búsqueda en la base de la tabla users utilizando el ID, si el usuario se encuentra mandamos un mensaje a consola mostrando el ID y nombre del usuario conectado. Posteriormente con `socket.emit` mandamos los parámetros del usuario.

```
if(req.session.userID != null){      //Al iniciar sesión...  
  db.query("SELECT * FROM users WHERE id=?", [req.session.userID], function(err, rows, fields){  
    console.log('Sesión iniciada con el UserID: ' + req.session.userID + ' Y nombre de usuario: ' +  
              req.session.Username); //Mostrar mensaje con las propiedades del usuario conectado  
    socket.emit("logged_in", {user: req.session.Username, email: req.session.correo, room: req.session.roomName});  
  });  
} else{  
  console.log('No hay sesión iniciada'); //Mostrar mensaje de que no se ha iniciado la sesión  
}
```

Si el usuario no se encuentra mandamos un mensaje a la consola.

```
socket.on("login", function(data){  
  const user = data.user,  
        pass = data.pass;  
  roomID = data.roomID;  
  roomName = data.roomName;
```

Con `socket.on` Recibimos los parámetros y los igualamos a `data`, (nombre parámetro) como se muestra a continuación.

A continuación, hacemos una búsqueda en la bd tabla usuarios utilizando el nombre del usuario si no existe muestra en consola un mensaje de error, de lo contrario realizamos el siguiente código.

```
else{  
  console.log(rows); //Imprime la información del usuario conectado en consola  
  
  const dataUser = rows[0].Username, //Se crean nuevas variables del usuario  
        dataPass = rows[0].Password,  
        dataCorreo = rows[0].email;  
  
  if(dataPass == null || dataUser == null ){  
    socket.emit("error");  
  }  
  if(user == dataUser && pass == dataPass){  
    console.log("Usuario correcto!"); //Mostrar mensaje de verificación  
    socket.emit("logged_in", {user: user, email: dataCorreo, room: roomName, roomID: roomID}); //Manda los param  
    req.session.userID = rows[0].id; //Asignamos los parametros en sesión  
    req.session.salaID = rows[0].id;  
    req.session.Username = dataUser;  
    req.session.correo = dataCorreo;  
    req.session.roomID = roomID;  
    req.session.roomName = roomName;  
    req.session.save(); //Guarda  
    socket.join(req.session.roomName); //pasamos de nombre de sesión el nombre de la sala (crear sala con join)  
    socket.emit('armadoHistorial');  
    bottxt('entroSala'); //mandar el parametro especificado  
  } else{  
    socket.emit("invalido");  
  }  
}
```

Realizamos 6 `Socket.on` para:

- ❖ Mostrar el historial de mensajes

Buscamos en la base de datos tabla mensajes los mensajes guardados y con un `Socket.emit` mandamos los parámetros de mensajes.

```
socket.on('historial', function(){
  console.log('Buscando historial de la sala: ' + req.session.roomName);

  db.query('SELECT s.nombre_sala, u.Username, m.mensaje FROM mensajes m INNER JOIN salas s ON s.id = m.sala_id INNER JOIN users u ON u.id = m.user_id WHERE m.sala_id = ' + req.session.roomID + ' ORDER BY m.id ASC', function(err, rows, fields){
    socket.emit('armadoHistorial', rows);
  });
});
```

- ❖ Agregar un nuevo usuario (Registrar)

Recibimos los parámetros y creamos nuevas variables. Al introducir la información en la caja de texto si no está vacía agregamos un nuevo usuario en la base de datos con la consulta “`INSERT INTO`” en la tabla users y mandamos a consola un mensaje de verificación.

De lo contrario mandamos los parámetros de “vacío” con un `socket.emit`.

```
//Función para agregar un usuario
socket.on('addUser', function(data){ //Se reciben los parametros del nuevo usuario
  const user = data.user,
  pass = data.pass,
  email = data.email;

  if(user != "" && pass != "" && email != ""){ //Verificamos que no vengan vacios
    console.log("Registrando el usuario: " + user); //Si no estan vacios,agregamos un nuevo usuario en la base de datos
    db.query("INSERT INTO users('Username', 'Password', 'email') VALUES(?, ?, ?)", [user, pass, email], function(err, result){
      if(!err)
        throw err;

      console.log(result); //Imprimimos el resultado

      console.log("Usuario " + user + " se dio de alta correctamente!"); //Mandar mensaje de verificación
      socket.emit('UsuarioOK');
    });
  }else{
    socket.emit('vacio'); //Manda a vacio
  }
});
```

- ❖ Cambiar de salas

```
socket.on('cambiodesala', function(data){

  const idSala = data.idSala,
  nombreSala = data.nombreSala;

  socket.leave(req.session.roomName);

  req.session.roomID = idSala;
  req.session.roomName = nombreSala;

  socket.join(req.session.roomName);
  bottxt('cambioSala');

});
```

- Electrónica
- Cocina

Recibimos los parámetros de las salas creamos nuevas variables.

Hacemos un `socket.leave` y un `socket.join`. Llamamos al bot de cambio sala.

❖ Crear un nuevo mensaje

Hacemos una consulta en la base de datos para crear nuevos mensajes e introducirlos en la tabla mensajes, mandar un mensaje de verificación en consola. Asimismo mandamos los parámetros de "mensaje" con un `socket.emit`.

```
socket.on('mjsNuevo', function(data){ // Función para crear el mensaje nuevo.

    //const sala = 0; // definimos el id de la sala para posterior función.

    db.query("INSERT INTO mensajes(`mensaje`, `user_id`, `sala_id`, `fecha`) VALUES(?, ?, ?, CURDATE())", [data, req.session.userID, req.session.roomID], function(err, result){
        if(!err)
            throw err;

        console.log(result);      //Imprimimos resultado

        console.log('Mensaje dado de alta correctamente!');

        socket.broadcast.to(req.session.roomName).emit('mensaje', {      //Mandamos info a mensaje
            usuario: req.session.Username,           //Le enviamos quien envió el mensaje
            mensaje: data
        });

        socket.emit('mensaje', {
            usuario: req.session.Username,
            mensaje: data
        });
    });
});
```

❖ Buscar las salas existentes

Buscamos en la base de datos tabla salas las salas existentes y con un `Socket.emit` mandamos los parámetros de salas.

```
socket.on('getSalas', function(data){
    db.query('SELECT id, nombre_sala FROM salas', function(err, result, fields){
        if(err) throw err;
        socket.emit('Salas', result);
    });
});
```

❖ Salir del Chat.

```
socket.on('salir', function(request, response){
    req.session.destroy();
});
```

Recibimos los parámetros de salir y destruimos sesión.

Por último, hacemos una función de "bot", se utilizarán 2 mensajes con este bot.

- ❖ Mostrar que el usuario entro a una sala
- ❖ Mostrar que el usuario cambio de sala

```
function bottxt(data){
    entroSala = 'Bienvenido a la sala ' + req.session.roomName;
    cambioSala = 'Cambiaste de sala a ' + req.session.roomName;
    /* sefue = 'El usuario ' + req.session.Username + ' ha salido' */

    if(data == "entroSala"){
        socket.emit('mensaje',{
            usuario: nameBot,
            mensaje: entroSala
        });
    }

    if (data == "cambioSala"){
        socket.emit('mensaje',{
            usuario: nameBot,
            mensaje: cambioSala
        });
    }
};
```

✓ index.html

En el archivo .html como primer paso creamos un login e introducimos dos cajas de texto para iniciar sesión:

- ❖ Nombre del usuario
- ❖ Contraseña

```
<main class="form-signin">
  <h1 class="h3 mb-3 fw-normal">Iniciar sesión</h1>    <!-- Creamos el login -->

  <div class="form-floating">
    <input type="text" class="form-control" id="userName" placeholder="name@example.com" name="username">
    <label for="floatingInput">Nombre de usuario</label>
  </div>
  <div class="form-floating">
    <input type="password" class="form-control" id="Password" placeholder="Password" name="password">
    <label for="floatingPassword">Contraseña</label>
  </div>
```

Asimismo, agregamos una caja de texto de selección opcional con el nombre de las 2 salas disponibles:

```
<h5 class="h5 ">Salas Disponibles: </h5>
<h8 class="h8 ">-Clase de Electrónica</h8>
<br>
<h8 class="h8 mb-3 fw-normal">-Clase de Cocina</h8>
<br>
<div class="form-floating">
  <select class="form-select" name="rooms" id="rooms">
    <option selected>Seleccionar sala</option>
  </select>
</div>
```

- ❖ Electrónica
- ❖ Cocina

Por ultimo agregamos dos botones

- ❖ Entrar
- ❖ Registrar

```
<button class="w-100 btn btn-lg btn-primary" type="button" id="Login">Entrar</button>
<button class="w-100 btn btn-lg btn-warning" type="button" id="registrar" data-toggle="modal" data-target="#registro">Registrar</button>
<p class="mt-5 mb-3 text-muted">&copy; 2021</p>
</main>
```

Como segundo paso mostramos un mensaje de bienvenida con el nombre y correo del usuario conectado.

```
<div id="wrapper" style="display: none;">    <!-- Mostramos un mensaje de bienvenida tomando el nombre introducido obtenido de la base de datos -->
  <div id="menu">
    <p> Sala: <b id="SalaNombre"></b></p>
    <p class="bienvenido"> Conectado como <b id="usernameTag"></b>, con correo: <b id="emailUser"></b></p>
    <p class="logout"><a id="exit" href="/">Salir del chat</a></p>
  </div>
```

Posteriormente creamos la caja del chat:

```
<!-- Caja del chat que contendrá todos los mensajes. -->
<div id="chatbox">
</div>

<input name="usermsg" type="text" id="mensaje" size="63"/>
<input type="button" name="submitmsg" id="enviarMensaje" value="Enviar Mensaje"/>
<select class="form-control" name="roomsCambio" id="roomsCambio">
    <option selected>Cambiar de sala</option>
</select>

</div>
```

Como tercer paso creamos el registro con 3 cajas de texto

- ❖ Nombre del usuario
- ❖ Contraseña
- ❖ Correo

Y agregamos los botones de

- ❖ Cerrar
- ❖ Registrar

```
<!-- Modal -->
<div class="modal fade" id="registro" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">
<div class="modal-dialog" role="document">
    <div class="modal-content">
        <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel">Registro</h5>
            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
        <div class="modal-body">
            <div class="form-floating">
                <input type="text" class="form-control" id="userNameR" placeholder="name@example.com" name="username" required>
                <label for="floatingInput">Nombre de usuario</label>
            </div>
            <div class="form-floating">
                <input type="password" class="form-control" id="PasswordR" placeholder="Password" name="password" required>
                <label for="floatingPassword">Contraseña</label>
            </div>
            <div class="form-floating">
                <input type="email" class="form-control" id="correo" placeholder="correo" name="correo" required>
                <label for="floatingPassword">Correo</label>
            </div>
        </div>
        <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-dismiss="modal">Cerrar</button>
            <button type="button" class="btn btn-primary" id="sendResgistro">Registrar</button>
        </div>
    </div>
</div>
</div>
```

Pasamos al socket y mandamos `getSalas` con un `socket.emit`, y recibimos los parámetros de Sala con un `socket.on` para poder entrar o cambiar de sala. Obtenemos las salas con su ID y con la selección de opción.

```
<script>
$(document).ready(function(){

    var socket = io();
    let salas = [];

    socket.emit('getSalas');

    socket.on('Salas', function(data){
        $.each(data, function(id, val){
            $('#rooms').append($('', {
                value: data[id].nombre_sala,
                text: data[id].nombre_sala,
                id: data[id].id
            }));
        });
    });
});
```

```
$('#roomsCambio').append($('', {
    value: data[id].nombre_sala,
    text: data[id].nombre_sala,
    id: data[id].id
}));
});
```

```
$('#roomsCambio').change(function(){
    //roomID = $(this).val();
    roomID = $(this).find('option:selected').attr('id');
    roomName = $(this).find('option:selected').text();

    /*roomID = roomID + 1;*/

    $("#SalaNombre").text(roomName);
    $("#chatbox").empty();

    socket.emit('cambiodesala', {
        idSala: roomID,
        nombreSala: roomName
    });
    socket.emit('historial');
    console.log('cambio select a ID: ' + roomID + 'con nombre ' + roomName);
});
```

Posteriormente pasamos a la función de Login, al entrar mandamos los parámetros con un `socket.emit` y asignamos los valores.

```
$("#Login").click(function(){  //Al entrar en login se mandan los parametros
    socket.emit("login", {
        user: $("#userName").val(),  //Asignar valores
        pass: $("#Password").val(),
        roomID: $("#rooms").find('option:selected').attr('id'),
        roomName: $("#rooms").find('option:selected').text()
    });
});
```

Para el registro cuando agregamos un nuevo usuario se mandan los parámetros del nuevo usuario

```
$("#sendRegistro").click(function(){
    socket.emit("addUser", {
        user: $("#userNameR").val(),
        pass: $("#PasswordR").val(),
        email: $("#correo").val(),
    });
});
```

- ❖ Nombre
- ❖ Contraseña
- ❖ Correo

Para la función de salir de la sesión mandamos los parámetros de salir.

```
$(".logout").click(function(){
    socket.emit("salir");
});
```

Como siguiente paso realizamos la función de "enviarMensaje". Si no hay un mensaje escrito mandamos un mensaje de alerta.

De lo contrario al escribir el mensaje se le asigna un valor a la variable y mandamos los parámetros del nuevo mensaje con un `socket.emit`

```
$('#enviarMensaje').click(function(){
    if($("#mensaje").val().length <= 0){ //Si no hay
        alert("Escribe el mensaje para poderlo enviar.");
    }else{
        var mensaje = $('#mensaje').val() //Si se escribe
        socket.emit('mjsNuevo', mensaje); // Enviamos el m
    }
});
```

Realizamos 7 `Socket.on` para:

- ❖ Recibir los parámetros de "logged_in"

Le asignamos a `usernameTag` el valor de texto de `data.user` para al decir "Bienvenido" aparezca el nombre del usuario conectado, asimismo con el correo y sala con sus parámetros correspondientes.

```
socket.on("logged_in", function(data){
    console.log(data);
    $(".form-signin").hide();
    $("#wrapper").show();
    $('#usernameTag').text(data.user); /
    $('#emailUser').text(data.email); /
    $('#SalaNombre').text(data.room); /
    socket.emit('historial');
});
```

- ❖ Indicar que un usuario tiene contraseña incorrecta

```
socket.on("invalido", function(){
  alert("Usuario y/o contraseña incorrectos.");
});
```

- ❖ Mostrar un error

```
socket.on("error", function(){
  alert("Error: Intenta de nuevo!");
});
```

- ❖ Indicar que los campos son vacíos

```
socket.on("vacio", function(){  //Recibe la información de que los campos estan vacios
  alert("Error: Llena todos los campos.!"); //Muestra un mensaje de alerta
});
```

- ❖ Dar de alta un usuario en el registro

```
socket.on("UsuarioOK", function(){  //Recibe
  $('#registro').modal('hide');    //Esconder la modal
  alert("Dado de alta correctamente.");
});
```

- ❖ Escribir un mensaje

Esta función tiene de respuesta el nuevo mensaje, donde concatenamos e insertamos en la caja del chat.

```
socket.on('mensaje', function(data){
  if(data.usuario == "BotChat"){
    var nuevoMensaje = '<small class="bot"><b>' + data.usuario + ' -</b> ' + data.mensaje + '</small>';
  }else{
    var nuevoMensaje = '<span class="mensajeEnviado"><b>' + data.usuario + ' dice: </b>' + data.mensaje +
  }
  $('#chatbox').append(nuevoMensaje + '<br>');
  $('#mensaje').val("");
});
```

❖ Mostrar el historial de mensajes

```
socket.on('armadoHistorial', function(data){
    var historial = "";
    $.each(data, function(id, val){
        historial += '<p class="mensajeEnviado"><b>' + data[id]['Username'] + 
'dijo:</b> ' + data[id]['mensaje'] + '</p>';
    });
    historial += '<small class="bot"><b> BotChat -</b> Ultimos mensajes del historial de la sala</small>';

    $('#chatbox').append(historial + '<br>');
});

});
```

✓ **chat.sql**

En el archivo .sql creamos una base de datos con la consulta “`create database chat`” y creamos 3 tablas:

❖ **table mensajes**

```
-- -----
-- Table structure for mensajes

DROP TABLE IF EXISTS `mensajes`; /*Si existe una tabla con ese nombre eliminarla, si no crearla*/
CREATE TABLE `mensajes` (
  `id` int unsigned NOT NULL AUTO_INCREMENT,
  `mensaje` text CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `user_id` int NOT NULL,
  `sala_id` int NOT NULL,
  `fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb3 COLLATE=utf8_spanish_ci ROW_FORMAT=DYNAMIC;
```

❖ **table users**

```
-- -----
-- Table structure for users

DROP TABLE IF EXISTS `users`; /*Si existe una tabla con ese nombre eliminarla, si no crearla*/
CREATE TABLE `users` (
  `id` int NOT NULL AUTO_INCREMENT,
  `Username` varchar(30) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `Password` varchar(60) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `email` varchar(30) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC;
```

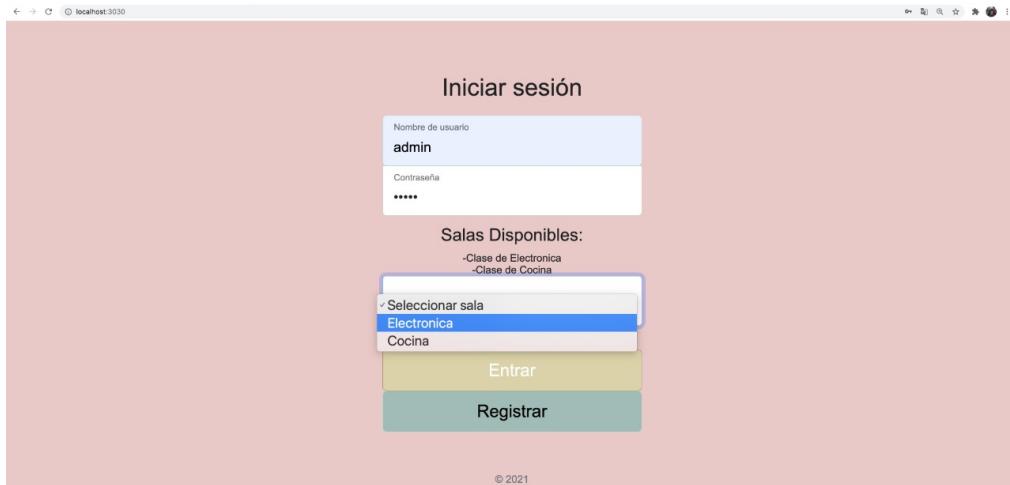
❖ table salas

```
-- 
-- Table structure for salas 

DROP TABLE IF EXISTS `salas`; /*Si existe una tabla con ese nombre eliminarla, si no crearla*/
CREATE TABLE `salas` (
  `id` int NOT NULL AUTO_INCREMENT,
  `nombre_sala` varchar(30) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `fecha_creación` date NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC;
```

✓ ¡RESULTADOS!

INICIAR SESIÓN



CHAT



CAMBIAR DE SALA



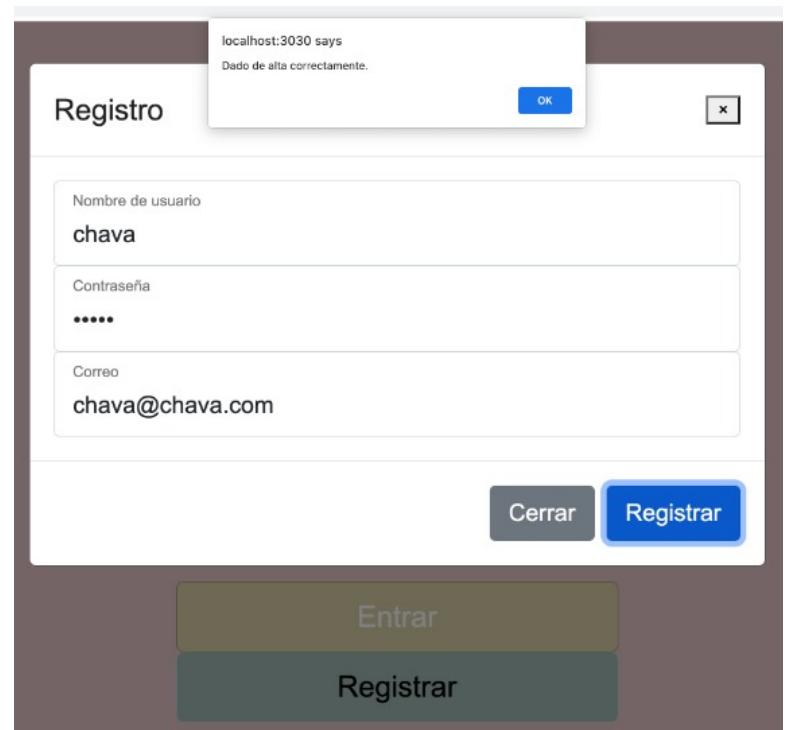
MANDAR UN NUEVO MENSAJE Y MOSTRARLO EN EL HISTORIAL



MANDAR MENSAJE DE ERROR



REGISTRAR NUEVO USUARIO.



GLOSARIO:

- Base de Datos

Colección organizada de información estructurada

- Cookies

Archivo creado por un sitio web que contiene pequeñas cantidades de datos y que se envían entre un emisor y un receptor.

- Consulta

Método para acceder a los datos en las bases de datos

- Enrutamiento

Función de buscar un camino entre todos los posibles en una red de paquetes cuyas topologías poseen una gran conectividad.

- Full-duplex

Término utilizado para definir a un sistema que es capaz de mantener una comunicación bidireccional, enviando y recibiendo mensajes de forma simultánea.

- Localhost

Nombre reservado que tienen todas las computadoras, routers o dispositivos independientemente de que dispongan o no de una tarjeta de red ethernet

- Node.js

Entorno en tiempo de ejecución multiplataforma de javascript, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos.

- Servidor

Programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.

CONCLUSIÓN:

En conclusión, ya que el trabajo lo fuimos trabajando a lo largo de la segunda parcial, fue más sencillo comprenderlo ya que fuimos adquiriéndola información de a poco, dividir las actividades y finalmente crear un chat funcional con todos los elementos que incluimos nos ayudó a comprender mejor como funciona cada cosa e ir haciendo comentarios fue muy esencial para recordar que se hace en cada apartado del código.

Aprendí que socket es muy útil para comunicar un cliente y un servidor lo cual desarrollamos en nuestro proyecto al crear un chat que podía comunicarse con distintos usuarios.

Link repositorio:

<https://github.com/daquin0/Portafolio>