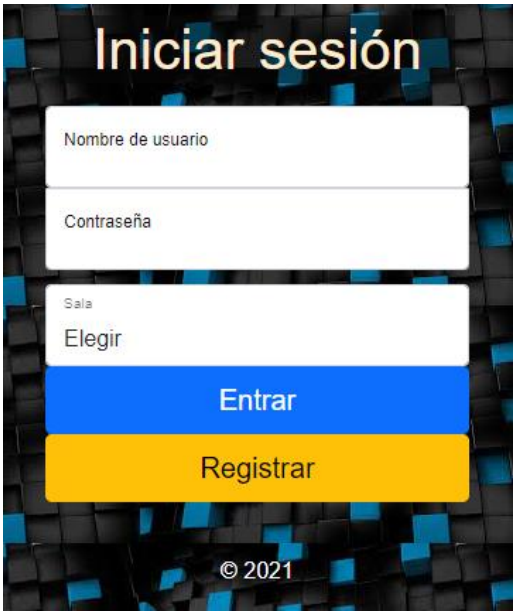
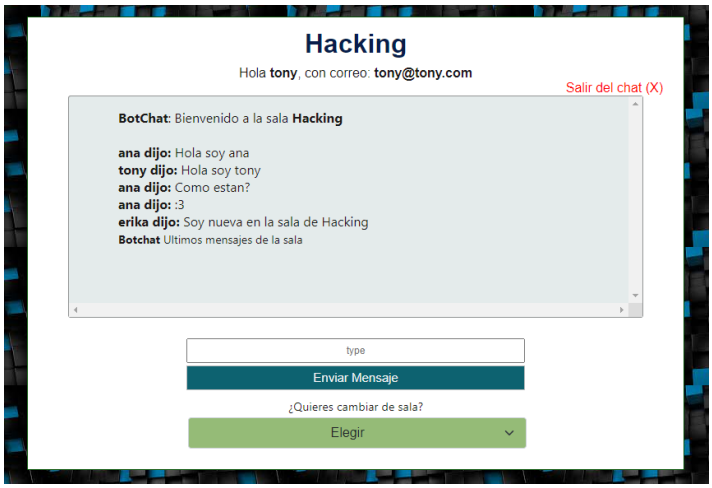
UNIVERSIDAD DE COLIMA

FACULTAD DE TELEMÁTICA

ING. EN TECNOLOGÍAS DE INTERNET

“CHAT CON SESIONES Y SALAS”





PROGRAMACIÓN DISTRIBUÍDA

RAMÍREZ GARCÍA CARLOS ANTONIO

PROF. MONTAÑO ARAUJO SERGIO ADRIÁN

4°C

25/JUNIO/2021

ÍNDICE

INTRODUCCIÓN

Esta documentación pertenece a un proyecto de chat, donde los usuarios pueden registrarse fácilmente y acceder a las distintas salas de chat para interactuar con otros usuarios que estén conectados a la misma sala. Dicho proyecto está elaborado con el lenguaje de programación Javacript, HTML y con el entorno de ejecución NodeJs.

El objetivo principal de este proyecto es crear la comunicación en tiempo real a través de salas de chat, manejando sesiones de usuario y cookies, haciendo un entorno amigable para el usuario, pues la usabilidad del sitio es muy sencilla y cuenta con muy pocas restricciones para tener acceso.

En esta documentación se explica la forma en la que fue desarrollada esta aplicación web a nivel de código y las tareas que realizan cada una de las funciones. También se explica el manejo de datos y como se transmiten del cliente al servidor y viceversa.

INSTALACIÓN DE DEPENDENCIAS Y LIBRERÍAS

Una vez creado nuestro directorio donde vamos a almacenar nuestro proyecto abrimos la terminal y nos dirigimos al directorio. Ya que estamos ahí ejecutamos el comando ***npm init*** para inicializar nuestro proyecto con NodeJS. Una vez ejecutado se crean las carpetas package.json, node_modules y package-lock.json. Por último, instalamos las siguientes dependencias:

- *Npm cookie-parser.*
- *Npm express.*
- *Npm express.session.*
- *Npm mysql*
- *Npm nodemon*
- *Npm socket.io*

Las dependencias instaladas aparecen en el archivo *package.json* en la sección "dependencies".

SERVIDOR

Nuestro servidor se encuentra en el archivo llamado "index.js", aquí se crean todas las funciones que va a consumir nuestro sitio del chat. Comencemos a explicar su contenido:

DECLARAR DEPENDENCIAS

Comenzamos declarando las dependencias que vamos a utilizar, almacenándolas en su respectiva variable. Y creamos la variable *"nameBot"* para asignarle el nombre al *bot* que emitirá notificaciones en el chat.

```
const express= require ('express'),
socket= require('socket.io'),
mysql= require('mysql'),
cookieParser= require('cookie-parser'),
session= require('express-session');
```

```
var app= express();
const nameBot= 'BotChat';
```

ARRANCAR SERVIDOR

Se inicializa el servidor y se almacena en la variable llamada "*server*", el servidor estará corriendo en el puerto **3030**, y notificamos en la consola cuando está corriendo el servidor.

```
var server= app.listen(3030, ()=>{
    console.log("Servidor trabajando en el puerto
    3030");
})
```

CONFIGURACIÓN DE SOCKET Y EXPRESS

Declaramos la variable "*io*" para crear la conexión con socket.io.

```
var io= socket(server);
```

Ahora creamos la variable "*sessionMiddleware*" para guardar el **id** de la sesión en el lado del servidor.

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

Creamos una función para que socket maneje por medio de **sessionMiddleware** las **request** que recibe el servidor y las **response** que emite.

```
io.use(function(socket,next){
    sessionMiddleware(socket.request, socket.request.
    res, next);
})
```

Le decimos a la variable "*app*" que maneje la variable "*sessionMidleware*" para que ejecute su función. Y también que maneje la dependencia **"cookie-parse"**

```
app.use(sessionMiddleware);
app.use(cookieParser());
```

Para la URL de nuestro sitio, se creó una ruta estática de la siguiente forma.

```
app.use(express.static('./'));
```

CONEXIÓN DE BASE DE DATOS

Se crea la conexión con la base de datos, colocando la información que corresponde a la misma, en este caso, nos conectamos a la base de datos **'nodelogin'**. Y guardamos esa conexión en la variable **"db".** Después, verificamos que la conexión sea exitosa y cualquier notificación o error la imprimimos en la consola del servidor para enterarnos del estado de la conexión.

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

```
db.connect(function(err){
    if(!!err)
    throw err; //nos dice el error

    console.log('MySQL conectado: ' + config.host + ",
    usuario: " + config.user + ", Base de datos: " +
    config.base); //imprime que la conexion ha sido
    exitoSa, nos da el nombre del host, el usuario y
    la base de datos.
});
```

FUNCIONES DE SOCKET

Ahora creamos la función de socket **'connection'** para crear la conexión al servidor cada que un usuario entre al sitio. Dentro de esta función se crean todas las funciones que va a desempeñar el servidor y que serán consumidas por el sitio.

```
io.on('connection', function (socket) {
```

MOSTRAR SALAS

Se realiza una consulta donde se extraigan todas las salas disponibles en la base de datos, y se hace un socket.emit para exportar esas salas cuando se requiera.

```
db.query("SELECT * FROM salas", function(err,rows,
fields){
    const salas= rows;
    console.log(salas);
    socket.emit('showRooms',salas);
});
```

VERIFICAR ESTADO DE LA SESION

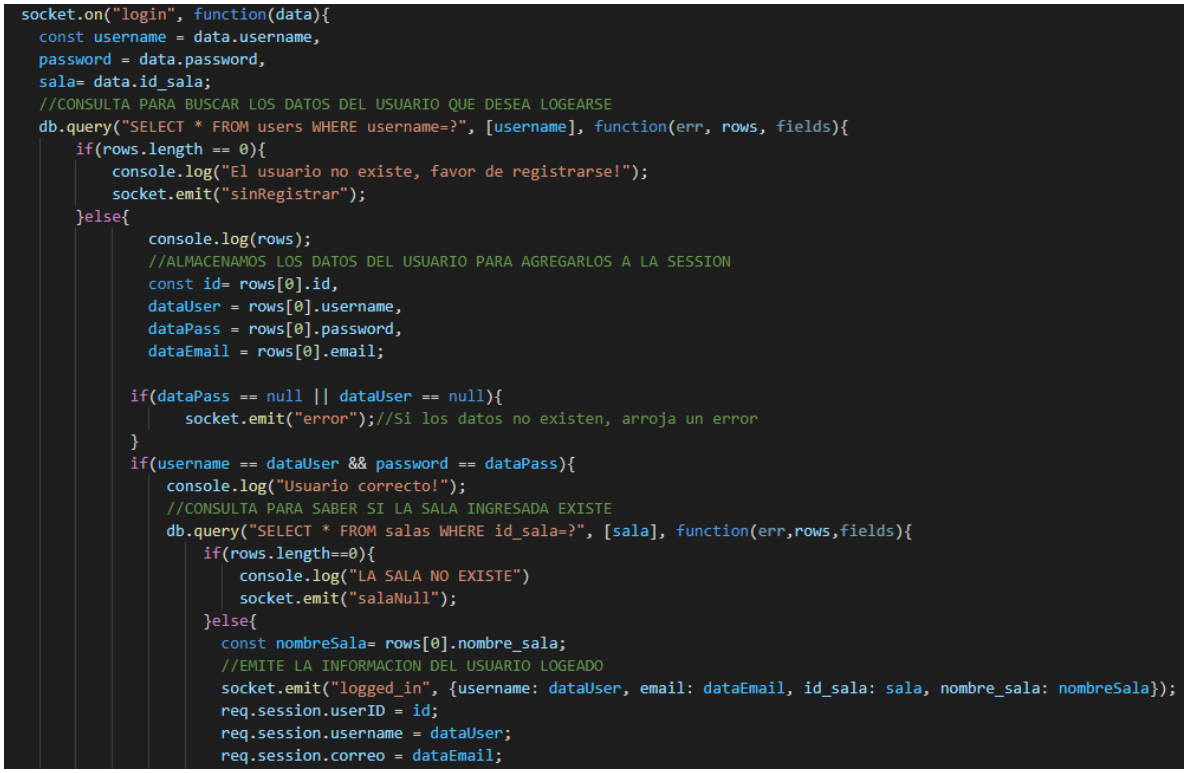
Verificamos el estado de la sesión, es decir, si el usuario ya esta 'logeado' o si aún no ha iniciado sesión. Si el usuario esta logeado lo notificamos en la consola y emitimos un socket.emit que contiene la información necesaria para crear la sesión. En caso contrario, se notifica en la consola que no existe una sesión iniciada.

```
if(req.session.userID != null){
    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);
        socket.emit("logged_in", {user: req.session.
        username, email: req.session.correo});
    });//SI EL USUARIO YA INICIÓ SESION CORRECTAMENTE NOS
    NOTIFICA LOS DATOS DE QUIEN SE HA LOGEADO
}else{
    console.log('No hay sesión iniciada'); //MUESTRA ESTE
    MENSAJE SI EL USUARIO NO HA INICIADO SESION
}
```

FUNCIÓN DE LOGIN

Esta función permite realizar o rechazar el acceso al chat. Recibe la información de parte del cliente cuando intenta logearse, después analiza la información recibida haciendo una consulta para saber si el usuario esta registrado, si no está registrado, le notifica que debe registrarse, si está registrado ahora hace otra consulta para verificar que la sala seleccionada esté disponible. Si la sala no está disponible, se le notifica y se le sugiere seleccionar otra, si existe la sala junto con los demás datos del usuario, se le da acceso al chat.

Se hace un ***socket.emit('logged_in'***) para enviar la información del usuario para que sea almacenada en la sesión. Se guarda la sesión con los datos del usuario y de la sala seleccionada. Se realiza un ***socket.join*** para indicar a qué sala se unió el usuario. Después llama a la función **'historial'** para que se muestren los mensajes de la sala y también se llama a la función **'notificacion'** para que el *botchat* agregué su mensaje para notificar el evento. Los console.log nos ayudan a ver en la consola si han ocurrido errores o si todo salió con éxito.



```
socket.on("login", function(data){
  const username = data.username,
  password = data.password,
  sala= data.id_sala;
  //CONSULTA PARA BUSCAR LOS DATOS DEL USUARIO QUE DESEA LOGEARSE
  db.query("SELECT * FROM users WHERE username=?", [username], function(err, rows, fields){
      if(rows.length == 0){
          console.log("El usuario no existe, favor de registrarse!");
          socket.emit("sinRegistrar");
      }else{
              console.log(rows);
              //ALMACENAMOS LOS DATOS DEL USUARIO PARA AGREGARLOS A LA SESSION
              const id= rows[0].id,
              dataUser = rows[0].username,
              dataPass = rows[0].password,
              dataEmail = rows[0].email;

            if(dataPass == null || dataUser == null){
                socket.emit("error");//Si los datos no existen, arroja un error
            }
            if(username == dataUser && password == dataPass){
                console.log("Usuario correcto!");
                //CONSULTA PARA SABER SI LA SALA INGRESADA EXISTE
                db.query("SELECT * FROM salas WHERE id_sala=?", [sala], function(err,rows,fields){
                    if(rows.length==0){
                        console.log("LA SALA NO EXISTE")
                        socket.emit("salaNull");
                    }else{
                      const nombreSala= rows[0].nombre_sala;
                      //EMITE LA INFORMACION DEL USUARIO LOGEADO
                      socket.emit("logged_in", {username: dataUser, email: dataEmail, id_sala: sala, nombre_sala: nombreSala});
                      req.session.userID = id;
                      req.session.username = dataUser;
                      req.session.correo = dataEmail;
```

```
                        req.session.id_sala= sala;
                        req.session.nombre_sala= nombreSala;
                        req.session.save(); //GUARDA LA SESION CON LOS DATOS DEL USUARIO
                        socket.join(req.session.nombre_sala); //INGRESA A LA SALA ELEGIDA
                        socket.emit('enviarHistorial'); //LLAMA A LA FUNCION PARA MOSTRAR HISTORIAL DE MENSAJES
                        Notificacion('LoginEnSala'); //NOTIFICACION EMITIDAS POR EL BOTCHAT
                        console.log(req.session);
                    }
                })
            }else{
                socket.emit("invalido"); //CUANDO LOS DATOS SON INCORRECTOS
            }
        }
    });
});
```

FUNCION ADDUSER

En la función **'adduser'** se recibe la información del usuario que intenta registrarse en nuestra base de datos. Primero se verifica que los campos estén completos (*username, password, email*), si alguno de los datos está vacío, se le notifica que es obligatorio llenar todos los campos. Si los datos están completos realiza una consulta para validar lo siguiente:

a. Que el username no exista en algún otro usuario.
b. Que el email no exista en algún otro usuario.

Si esas condiciones se cumplen, entonces realiza una consulta ***'INSERT'*** para agregar al usuario a la base de datos **'nodelogin'** en la tabla **'users'** y se le notifica con una alerta que se ha registrado correctamente. De lo contrario, se le notifica que su *username o email* ya son existentes y que pruebe con otros datos.

```
socket.on('addUser', function(data){
    //ALMACENAMOS LOS DATOS QUE INGRESÓ EL USUARIO
    const user = data.user,
    pass = data.pass,
    email = data.email;

    if(user != "" && pass != "" && email != ""){
        console.log("Registrando el usuario: " + user);
        //CONSULTA PARA SABER SI EL USERNAME O EMAIL YA EXISTEN EN OTROS USUARIOS
        db.query("SELECT * FROM users WHERE username=? OR email=?",[user,email], function(err,rows,fields){
            if(rows.length>0){
                socket.emit("UserExistente");
            }else{
                //SI NO EXISTE EL USERNAME O EMAIL, AGREGA AL USUARIO A LA DB
              db.query("INSERT INTO users(`username`, `password`, `email`) VALUES(?, ?, ?)", [user, pass, email], function(err, result){
                  if(!!err)
                  throw err;//NOS DICE SI HAY UN ERROR
                  console.log(result);//IMPRIME RESULTADO DE LA CONSULTA
                  console.log('Usuario ' + user + " se dio de alta correctamente!.");
                  socket.emit('UsuarioOK');//NOS AVISA QUE EL USUARIO SE AGREGÓ CORRECTAMENTE
              });
            }
        })
    }else{
        socket.emit('vacio'); //NOS DICE QUE UNO O MAS CAMPOS ESTAN VACIOS
    }
});
```

FUNCION MENSAJE NUEVO

En la función **‘msjNuevo’** recibimos el mensaje que quiere enviar el usuario, junto con el id de la sala en la está conectado y el id del usuario quien envía el mensaje. Una vez que tenemos esos datos hacemos una consulta **‘INSERT’** para agregar el *mensaje, id de la sala, id del usuario, la hora y fecha.* Y se inserta a la base de datos **‘nodelogin’** en la tabla **‘mensajes’**.

Con la instrucción ***socket.broadcast.to ()*** enviamos el mensaje solamente a la sala en la que se envió y a todos los usuarios conectados a esa misma sala les aparecerá el nuevo mensaje en tiempo real.

Finalmente emitimos el mensaje y el nombre del usuario con un ***socket.emit(‘mensaje’).***

```
socket.on('mjsNuevo', function(data){ // FUNCION PARA CREAR
CADA MENSAJE NUEVO

    var sala= req.session.id_sala; // ID DE LA SALA
    var user= req.session.userID; //ID DEL USUARIO
    //CONSULTA PARA INSERTAR EL MENSAJE NUEVO A LA BD
    MENSAJES CON SU SALA A LA QUE CORRESPONDE
        db.query("INSERT INTO mensajes(`mensaje`, `user_id`,
        `sala_id`, `fecha`) VALUES(?, ?, ?, CURDATE())",
        [data, user, sala], function(err, result){
          if(!!err)
          throw err;//NOS NOTIFICA SI OCURRE UN ERROR.

          console.log(result);

          console.log('Mensaje dado de alta correctamente!.')
          ;
              //FUNCION QUE ENVIA EL MENSAJE UNICAMENTE A LA
              SALA DONDE CORRESPONDE
                  socket.broadcast.to(req.session.
                  nombre_sala).emit('mensaje',{
                      usuario: req.session.username,
                      mensaje: data
                  });

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

FUNCION SALIR

La función **‘salir’** funciona para cuando el usuario cierre sesión, se elimine la sala en la que está conectado y posteriormente destruye la session. Así la próxima vez que el usuario quiera entrar de nuevo al chat, será necesario logearse otra vez.

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

});
```

FUNCIÓN CAMBIO DE SALA

En la función **'cambioDeSala'** recibimos la sala a la que el usuario desea cambiarse. Después desconectamos al usuario de su sala actual con un ***socket.leave()***, para eso extraemos el nombre de la sala actual que se encuentra en la ***req.session.nombre_sala***.

Ahora actualizamos los datos de la sesión del usuario, ingresamos el id y el nombre de la nueva sala a la que se cambió. Y lo conectamos a esa sala con un ***socket.join***. Después hacemos una llamada a la función **'Notificacion'** para que el **botChat** muestre su mensaje. Por último, imprimimos en la consola del servidor la session del usuario y la sala a la que se ha cambiado, para comprobar que el cambio ha sido exitoso.

```
socket.on('cambioDeSala', function(data){
  const salaId= data.id_sala,
  nombreSala= data.nombre_sala;

  socket.leave(req.session.nombre_sala); //CIERRA LA SESION
  DE LA SALA ACTUAL
  //GUARDAMOS LOS DATOS DE LA NUEVA SALA EN SESSION
  req.session.id_sala= salaId;
  req.session.nombre_sala= nombreSala;
  socket.join(req.session.nombre_sala); //INGRESAMOS A LA
  NUEVA SALA
  Notificacion('CambioDeSala'); //NOTIFICA EL BOTCHAT QUE SE
  HA CAMBIADO DE SALA
  console.log(req.session, 'Ha cambiado a la sala: '+req.
  session.nombre_sala);
});
```

FUNCIÓN HISTORIAL

En la función **'historial'** se hace una consulta para extraer los mensajes de la sala con el id que se encuentra en ***req.session.id_sala***. Una vez hecha esa consulta se emiten los mensajes obtenidos de la consulta con un ***socket.emit***.

```
socket.on('historial',function(){
  console.log('Creando historial');
  //CONSULTA PARA EXTRAER LOS MENSAJES DE LA SALA
  db.query('SELECT nombre_sala as sala, username, mensaje
  FROM mensajes INNER JOIN salas ON id_sala=sala_id INNER
  JOIN users on id=user_id WHERE sala_id='+req.session.
  id_sala+' order by id_mensaje ASC', function(err,rows,
  fields){
      socket.emit('enviarHistorial', rows);
  });
})
```

FUNCIÓN NOTIFICACIONES

En la function **Notificaciones** creamos los mensajes que va a emitir e ***botchat*** cuando se realicen eventos como: ingreso a una sala y cambio de sala.

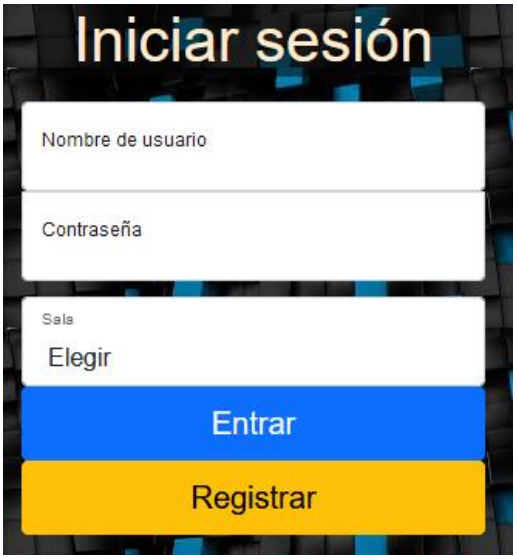
```
  function Notificacion(evento){
      const LoginEnSala= 'Bienvenido a la sala <b>'+req.
      session.nombre_sala+'</b>',
      CambioDeSala= 'Cambiaste a la sala <b>'+req.session.
      nombre_sala+'</b>';

      if(evento=="LoginEnSala"){
          socket.emit('notificacion',{
              usuario: nameBot,
              mensaje: LoginEnSala
          })
      }

      if(evento=="CambioDeSala"){
        socket.emit('notificacion',{
            usuario: nameBot,
            mensaje: CambioDeSala
            })
       }
  }
});
```

VISTAS HTML

VISTA LOGIN

La vista HTML está compuesta de un formulario para el **login**, es la vista principal, es decir, aparece cuando entras al sitio.

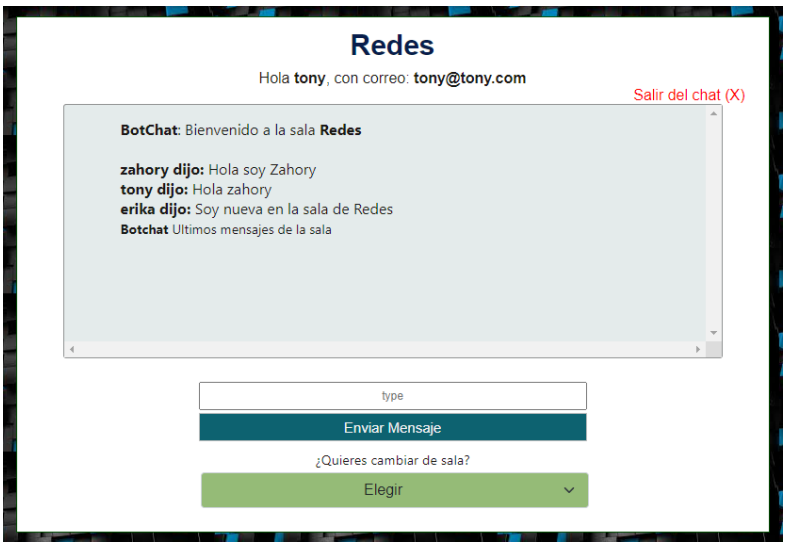
```
<main class="form-signin">
    <h1 class="h1 mb-3 fw-normal">Iniciar sesión</h1>
    <div class="form-floating">
      <input type="text" class="form-control" id="userName"
      placeholder="name@example.com" name="username">
      <label for="floatingInput">Nombre de usuario</label>
    </div>
    <div class="form-floating">
      <input type="password" class="form-control" id="Password"
      placeholder="Password" name="password">
      <label for="floatingPassword">Contraseña</label>
    </div>
    <div class="form-floating">
      <select type="text" name="idsala" id="salas"
      class="form-control" placeholder="idsala"></select>
      <label for="floatingsala">Sala</label>
    </div>

    <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  id="year">&copy; 2021</p>
</main>
```



VISTA DEL CHAT

Mostramos el nombre de la sala, el usuario que está conectado, el cuadro para el chat donde se muestran los mensajes, un input para escribir el mensaje y su botón para enviarlo, un enlace para cerrar sesión por último un select para que cambie de sala cuando lo desee.

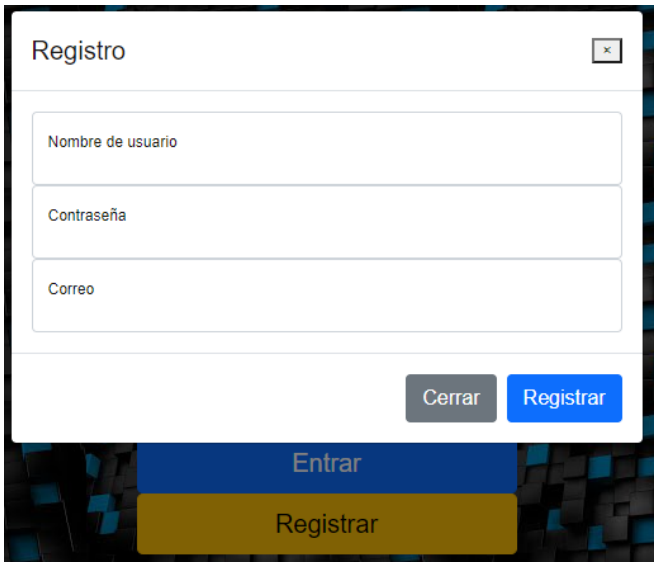
```
<!--VISTA DEL CHAT-->
<div id="wrapper" style="display: none;">
  <div id="menu">
    <h2><b id="nombresala"></b></h2>
    <p class="bienvenido"> Hola <b id="usernameTag"></b>, con
    correo: <b id="emailUser"></b> </p>
    <p class="logout"><a id="exit" href="/">Salir del chat (X)
    </a></p>
  </div>

  <div id="chatbox" style="overflow: scroll;" >
<!--        Caja del chat que contendrá todos los mensajes. -->
  </div>

    <input name="usermsg" type="text" id="mensaje" size="63"
    placeholder="type" style="text-align: center;"/>
    <input type="button" name="submitmsg" id="enviarMensaje"
    value="Enviar Mensaje"/>
    <p class="question">¿Quieres cambiar de sala?</p>
    <select class="form-select" name="cambioSala"
    id="cambioSala">
    </select>
  </div>
```



VISTA DEL REGISTRO

Esta vista es una caja emergente que aparece cuando el usuario da click en el botón registrar de la vista del login, muestra los campos que debe ingresar el usuario para su registro, como username, password y email.

```html
<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>
```



HEAD HTML

En este apartado pusimos los links que se conectan a Bootstrap para poder crear el estilo del sitio.

Y configuramos para suministrar información codificada a navegadores, motores de búsqueda sobre una página web y la gestión del contenido de una página web.

```html
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
    Bootstrap contributors">
    <meta name="generator" content="Hugo 0.83.1">
    <title>Salas de chat</title>

    <link rel="canonical" href="https://getbootstrap.com/docs/5.0/
    examples/sign-in/">



    <!-- Bootstrap core CSS -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/
bootstrap.min.css" rel="stylesheet" integrity="sha384-
+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x"
crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/
bootstrap.bundle.min.js" integrity="sha384-gtEjrD/
SeCtmISkJkNUaaKMoLD0//ElJ19smozuHV6z3Iehds+3Ulb9Bn9Plx0x4"
crossorigin="anonymous"></script>
<meta name="theme-color" content="#7952b3">
```

ESTILOS

En este apartado creamos el diseño del sitio, medidas, colores, posiciones, márgenes y más.

```
    <style>
      html,
body {
  height: 100%;
  background-image:url(/bckgnd.jpg);
  background-size:auto;
}



body {
  display: flex;
  align-items: center;
  padding-top: 40px;
  padding-bottom: 40px;
  background-color: #f5f5f5;
}
```

SCRIPTS LIBRERIAS

En esta sección tenemos los scripts que se conectan a librerías como *jquery, socket.io, cloudfare, bootstrapcdn* para que el sitio admita las funciones que debemos ejecutar en el código para que el sitio funcione correctamente.

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.
9/umd/popper.min.js" integrity="sha384-ApNbgh9B
+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/
bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/
JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></
script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="/socket.io/socket.io.js"></script>
```

SCRIPT JAVASCRIPT

CONEXIÓN SOCKET

Declaramos la variable socket para crear la conexión con socket.io.

```
var socket = io();
```

MOSTRAR SALAS

Aquí hacemos un llamado a la función **'showRooms'** del servidor que nos devuelve las salas disponibles. Se recorre el vector que nos regresó la función del servidor y cada sala la guardamos en otro vector (**getSalas**) con formato *json*, ejemplo: *{id: 1, nombre_sala: redes}.*

Después de guardar todas las salas en el vector **getSalas** con formato *Json*, creamos la variable **infoSalas** donde se van a guardar las salas como *option*, le declaramos una option hidden para que funcione como un placeholder en el select. A continuación, recorremos el vector **getSalas** para crear las options que serán agregadas al select, en el value guardamos el id de la sala y después le ponemos el nombre de la sala que será la opción que aparecerá en el select.

Finalmente, la variable '**infoSalas'** que ahora ya contiene todas las opciones, la agregamos al select de la vista **login** y del chat con un innerHTML.

```
socket.on("showRooms",function(salas){
  var getSalas= new Array();
  $.each(salas, function(id,val){
    var info={id: salas[id].id_sala, nombre_sala: salas[id].
    nombre_sala}
    getSalas.push(info);
  });

  let infoSalas= '<option hidden selected>Elegir</option>';
  getSalas.forEach(sala=>{
    infoSalas+= `<option value=${sala.id}>${sala.nombre_sala}
    </option>`;
  });
  document.getElementById("salas").innerHTML= infoSalas; //
  AGREGAR SALAS AL SELECT DEL LOGIN
  document.getElementById("cambioSala").innerHTML=infoSalas;
  //AGREGAR SALAS AL SELECT DENTRO DEL CHAT
  console.log(getSalas);
});
```

BOTÓN LOGIN

Cuando el usuario haga click en el botón **Entrar** que se encuentra en la vista de *login*, se emite la información que ingresó el usuario en los campos a la función **'login'** del servidor para que autorice o rechace el acceso.

```
$("#Login").click(function(){
  socket.emit("login", {
    username: $("#userName").val(),
    password: $("#Password").val(),
    id_sala: $("#salas").val(),
    nombre_sala: $("#salas").find('option:selected').text()
  });
});
```

BOTÓN REGISTRAR

Cuando el usuario haga click en el botón '**Registrar'** que está en la vista del Registro, se emiten los datos ingresados en los campos a la función **'adduser'** del servidor para hacer la validación.

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

También se resetean los inputs.

```
$('#sendRegistro').click(function(){
  $("#userNameR").val("");
  $("#PasswordR").val("");
  $("#correo").val("");
});
```

CERRAR SESIÓN

Cuando el usuario haga click en el enlace ***'cerrar sesión'*** se llama a la función **'salir'** del servidor para cerrar la sesión y se emite una alerta dándole la despedida.

```
$(".logout").click(function(){
  socket.emit("salir");
  alert('Hasta la proxima');

});
```

ENVIAR MENSAJE

Cuando el usuario haga click en el botón **Enviar** en la vista del chat, se verifica que el mensaje no este vacío, si este vacío se le notifica que no puede enviar mensajes vacíos, si tiene contenido se extrae el mensaje del input y se manda a la función del servidor **'msjNuevo'** para generar el nuevo mensaje.

```
$('#enviarMensaje').click(function(){
  if($("#mensaje").val().length <= 0){
    alert("Escribe el mensaje para poderlo enviar.");//SI EL
    MENSAJE ESTA VACIO NOTIFICA EL ERROR
  }else{
    var mensaje = $('#mensaje').val()
    socket.emit('mjsNuevo', mensaje); // ENVIAMOS EL MENSAJE
    A LA FUNCION DE 'msjNevo' DEL SOCKET PARA CREAR EL NUEVO
    MENSAJE

  }
});
```

FUNCIÓN LOGGED_IN

Se hace un llamado a la función **'logged_in'** del servidor, ocultamos la vista del *login* y mostramos la vista del *chat*. Con los datos que nos devuelve agregamos en el título el nombre de la sala y en el párrafo de bienvenida el nombre y correo del usuario. Finalmente hacemos un llamado a la función **'historial'** del servidor para que muestre el historial de mensajes de la sala.

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

MOSTRAR HISTORIAL DE MENSAJES

Para mostrar el historial de los mensajes de la sala llamamos a la función **'enviarHistorial'** del servidor y recibimos todos los mensajes de la sala. Creamos una variable llamada **historial** y cuando recorramos el vector de los mensajes recibidos de la función de socket concatenamos cada mensaje con su formato visual en la variable historial. El *botchat* indica cuáles son los últimos mensajes de la sala y se concatena el contenido de la variable historial al cuadro del chat para mostrar los mensajes de la sala.

```
socket.on("enviarHistorial",function(data){
  var historial="";
  $.each(data, function(id,val){//CONCATENAMOS LOS MENSAJES
    historial+= '<p><b>'+data[id].username+' dijo: </b>'+
    data[id].mensaje+'</p>';
  })
  if(data!= null)
  historial+= '<small class="bot"><b>Botchat</b> Ultimos
  mensajes de la sala</small> <br>';//MENSAJE DEL BOTCHAT

  $('#chatbox').append(historial);//MOSTRAMOS EL HISTORIAL
  EN LA CAJA DEL CHAT
  console.log('mensajes');
  console.log("mensajes"+ data);
});
```

CONCATENAR NUEVO MENSAJE

Para concatenar el mensaje que envía el usuario llamamos a la función **'mensaje'** del ser servidor, nos devuelve el usuario y el mensaje, en una variable llamada **nuevoMensaje** se guarda el mensaje con el estilo en que se va a mostrar. Posteriormente concatenamos el nuevo mensaje a la caja del chat y reseteamos la caja del input donde se escriben los mensajes.

```
socket.on('mensaje', function(data){
    var nuevoMensaje = '<b>' + data.usuario + ' dice:</b> '
    + data.mensaje;
    $('#chatbox').append(nuevoMensaje + '</br>');
    $('#mensaje').val("");
  });
```

CAMBIO DE SALA

Cuando el usuario cambie de sala con el select que está en la vista del chat, se extrae el id y el nombre de la nueva sala, se vacía la caja donde se muestran los mensajes del chat y se cambia el título de la sala por el nombre de la sala nueva.

Finalmente se envían a la función **'cambiosDeSala'** del servidor los datos de la nueva sala para que haga el cambio y después se llama a la función **'historial'** del servidor para que muestre el historial de mensajes de la sala nueva.

```
$('#cambioSala').change(()=>{
 var idSala= $('#cambioSala').val(),
 nombreSala= $('#cambioSala').find('option:selected').text();

 $('#nombresala').text(nombreSala);
 $('#chatbox').empty();
//EMITE AL SOCKET LOS NUEVOS DATOS DE LA SALA A LA QUE SE
DESEA CAMBIAR
 socket.emit('cambioDeSala',{
   id_sala: idSala,
   nombre_sala: nombreSala
 });
 socket.emit('historial');//PARA ACTUALIZAR EL HISTORIAL
})

});
```

ERRORES Y NOTIFICACIONES

USUARIO INVÁLIDO

Cuando el usuario intente logearse y escriba mal sus datos se llama a la función **'invalido'** del servidor y se hace una alerta al usuario de que sus datos son incorrectos.

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

ERROR USERNAME Y PASSWORD NULL

Cuando el username y la password sean **null** se le notifica al usuario un error y se le pide intentarlo de nuevo.

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

ERROR DATOS VACIOS

Cuando el usuario no llene todos los campos del registro se manda una alerta diciendo que llene todos los campos.

```
socket.on("vacio", function(){
  alert("Error: Llena todos las campos.!");
});
```

REGISTRO CORRECTO

Cuando el registro de un usuario se realiza correctamente entonces le damos una alerta haciéndoselo saber.

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

ERROR USUARIO NO REGISTRADO

Si un usuario inserta datos inexistentes para logearse se llama a la función **'sinRegistrar'** del servidor y se le da una alerta diciendo que debe registrarse.

```
socket.on('sinRegistrar',function(){
  alert("Favor de registrarse");
});
```

ERROR SALA INEXISTENTE

Cuando el usuario ingrese una sala que no está disponible o es inexistente, se llama a la función **'salaNull'** del servidor y se le da una alerta diciendo que la sala no existe.

```
socket.on('salaNull',function(){
  alert("La sala NO existe");
});
```

ERROR USUARIO EXISTENTE

Si el usuario quiere registrarse con un nombre o correo que ya existen, se llama a la función **'UserExistente'** y se manda una alerta diciendo que el usuario o el email ya existen.

```
socket.on('UserExistente',function(){
  alert("El usuario/email ya existe");
});
```

NOTIFICACIONES DEL BOTCHAT

Para mostrar las notificaciones que manda el *botchat* llamamos a la función **'notificacion'** del servidor. Nos devuelve el mensaje que contiene la notificación. Creamos la variable **mensajeBot** para almacenar el mensaje en código HTML. Una vez creado el mensaje en código HTML lo concatenamos a la caja del chat. Y reseteamos el input donde se escribe el mensaje.

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

BASE DE DATOS

La base de datos fue creada en MySql y es llamada **"nodelogin",** se creó de la siguiente manera:

```
CREATE DATABASE IF NOT EXISTS `nodelogin` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

TABLA USERS

Se creó para almacenar la información de los usuarios, el comando se muestra a continuación:

```
CREATE TABLE users(id int(11) auto_increment,
username varchar(50) not null,
password varchar(255) not null,
email varchar(100) not null,
PRIMARY KEY (id));
```

TABLA SALAS

Se creó para almacenar las salas de chat disponibles.

```
CREATE TABLE salas (id_sala int auto_increment,
nombre_sala varchar(30)  not null,
fecha_creacion date not null,
PRIMARY KEY (id_sala));
```

TABLA MENSAJES

Se creó para almacenar la información de los mensajes que se envían en las salas de chat:

```
CREATE TABLE mensajes(id_mensaje int(11) 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_mensaje));
```

GLOSARIO

1. **Socket.io:** biblioteca que permite la comunicación en tiempo real, bidireccional y basada en eventos entre el navegador y el servidor.
2. **Express:** framework web más popular de Node, permite la escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes rutas URL.
3. **Express-session:**permite que los datos de la sesión se almacenan en el lado del servidor, acepta cookies.
4. **Dependencias:** usado en la Programación Orientada a Objetos, permite la creación de los objetos de manera práctica, útil, escalable y con una alta versatilidad del código.
5. **Librerias:** conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
6. **NodeJs:** entorno de ejecución de JavaScript orientado a eventos asíncronos, diseñado para crear aplicaciones network escalables.
7. **Directorio:** contenedor virtual donde se almacenan una agrupación de archivos informáticos y otros subdirectorios.
8. **Servidor:** software que devuelve información (páginas) cuando recibe peticiones por parte de los usuarios.
9. **Request:** representa una solicitud de respuesta (petición).
10. **Response:** representa la respuesta a una request (petición).
11. **Query:** sirve para extraer información de una base de datos.
12. **sala de chat:** lugar virtual donde los usuarios de chat, cada uno identificado por su nombre o nickname, se conectan para charlar con otros que se encuentran en la misma sala.
13. **Bootstrap:** framework front-end utilizado para desarrollar aplicaciones web y sitios mobile.
14. **Concatenar:** Unir cadenas, variables, etc.
15. **Botchat:** Interfaz de texto que notifica los eventos realizados por el usuario y muestra sus mensajes dentro de la caja de los mensajes del chat.

CONCLUSION

La implementación de este proyecto es de gran utilidad, pues permite la comunicación en tiempo real, la interacción que tiene con el usuario es muy amigable, además es muy sencilla y clara en sus instrucciones, alertas y notificaciones para que el usuario tenga una buena experiencia en el sitio.

Es eficaz para las ocasiones donde se quieres dividir grupos o comunidades de diálogos y hablar de un tema en especifico o simplemente para platicar entre amigos, además, una vez dentro del chat, te puedes cambiar a otras salas de forma rápida.

Todo eso es posible gracias al manejo de sesiones, cookies y socket.io que nos permiten crear sesiones de usuario y almacenarlas para tener “a la mano” la información del usuario, como su nombre, id o la sala en la que está conectado, de esa forma podemos ejecutar las funciones de forma más agilizada y estamos al pendiente en el lado del servidor de los movimientos que se realizan en el sitio.