



UNIVERSIDAD DE COLIMA
Facultad de Telemática



EDUCACIÓN CON
RESPONSABILIDAD
SOCIAL

UNIVERSIDAD DE COLIMA

UNIVERSIDAD DE COLIMA

Facultad de telemática

“Programación Distribuida”

Proyecto Chat-NodeJs-Socket.io

ALUMNA: MONDRAGÓN DELGADO MEZLY ZAHORY

PROFESOR: MONTAÑO ARAUJO SERGIO ADRIAN

GRADO Y GRUPO: 4 – C

25 de junio del 2021, Colima

INDICE

INTRODUCCIÓN.....	3
PROYECTO-CHAT HACIENDO USO DE SESIONES Y SALAS CON NODEJS.....	4
1. DESCRIPCIÓN DEL PROYECTO.....	4
1.1 OBJETIVOS.....	4
2. RECURSOS UTILIZADOS	4
2.1 DESCRIPCIÓN DE LOS RECURSOS UTILIZADOS	5
3. ESPECIFICACIÓN DE REQUERIMIENTOS.....	7
4. PROCEDIMIENTOS DE DESARROLLO	7
4.1 FUNCIONAMIENTO	7
DESARROLLO	7
4.2 DECLARACIÓN DE LIBRERÍAS, CONFIGURACIÓN DEL SERVER, RUTA Y DE LA BASE DE DATOS.....	7
4.3 LOGIN	9
4.4 REGISTRAR USUARIOS.....	15
4.5 CAMBIAR DE SALA	17
4.6 MENSAJES.....	19
4.7 HISTORIAL DE MENSAJES.....	20
4.8 BOT	22
4.9 CREACIÓN DE LA BASE DE DATOS	24
5. DISEÑO Y RESULTADOS	26
GLOSARIO.....	29
CONCLUSION.....	30

INTRODUCCIÓN

En el presente documento, se expondrá y hablará acerca de todo el desarrollo e implementación del proyecto que se desarrolló a lo largo del semestre de la segunda parcial en la materia de Programación Distribuida.

Se abordarán los temas desde la descripción del proyecto y en que consistió, así como los objetivos, los cuales básicamente fueron obtener como resultado un chat funcional, que hiciera uso de sesiones, incluyendo salas, así como el historial de los mensajes de cada una de ellas. También se hablará de las herramientas utilizadas, como las librerías que fueron requeridas, como express, express-session, cookie-parser, socket.io, etc.

Se explicará todo el proceso de desarrollo del chat, de cada una de las funciones que fueron creadas, así como todas las consultas necesarias para poder obtener buenos resultados del manejo de la base de datos. De igual manera, se muestran partes del código para entender cómo es que está implementado el chat y que tipo de funciones fueron requeridas, así como resultados de las pantallas finales del chat funcionando en el servidor de forma local.

PROYECTO-CHAT HACIENDO USO DE SESIONES Y SALAS CON NODEJS

1. DESCRIPCIÓN DEL PROYECTO

El proyecto-chat desarrollado a lo largo de la segunda parcial del semestre consta de un chat, con su login para ingresar a el, donde al ingresar podrá mandar mensajes y revisar el historial de mensajes de la sala a la que ingrese.

1.1 OBJETIVOS

- Este proyecto se realizó con el fin de aprender a como desarrollar e implementar un chat, mediante el uso de sesiones, así como a que los usuarios elijan la sala de chat a la que desean entrar y estén en comunicación con los demás usuarios que de igual forma están en esa sala.

2. RECURSOS UTILIZADOS

- Librerías
 - Socket.io
 - Express
 - Mysql
 - Cookie-parser
 - Express-Session
 - Nodemon
- Programas
 - Visual studio code
- Lenguajes
 - Javascript
 - HTML

2.1 DESCRIPCIÓN DE LOS RECURSOS UTILIZADOS

- **Librerías**

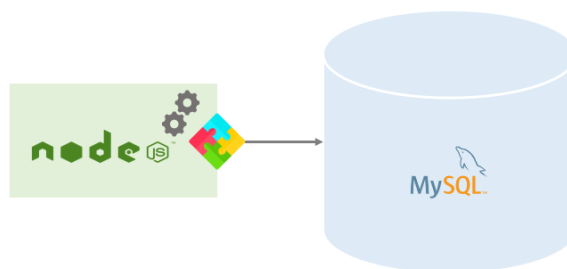
- **socket.io** → Permite la comunicación en tiempo real, bidireccional y basada en eventos entre el cliente y el servidor.



- **Express** → Framework web más popular de Node. Permite manejar peticiones del servidor con HTTP en diferentes rutas.



- **Mysql** → Implementar una base de datos en nuestro sistema web, haciendo consultas para administrar la base de datos.



- **Cookie-parser** → Es un middleware que permite hacer el manejo de sesiones mediante cookies.
- **Express-Session** → Middleware que almacena todos los datos de la sesión en el servidor. Es aquí donde toma los datos de la cookie que se mandó en la petición y solo guarda el id de esa cookie.

- **Nodemon** → Librería que se utiliza para guardar los cambios realizados en el servidor de manera automática.



- **Programas**

- **Visual studio code** → Entorno de editor de texto plano desarrollado por Microsoft totalmente gratuito y de código abierto para ofrecer a los usuarios una herramienta de programación avanzada como alternativa al Bloc de Notas.



- **Lenguajes**

- **Javascript** → Lenguaje de programación que funciona en los navegadores de forma nativa (lenguaje interpretado sin necesidad de compilación). Por tanto, se utiliza como complemento de HTML y CSS para crear páginas webs.



- **HTML** → Lenguaje que se utiliza para el desarrollo de páginas de Internet. Se trata de las siglas que corresponden a HyperText Markup Language.



3. ESPECIFICACIÓN DE REQUERIMIENTOS

- Contar con un server para manejar una base de datos de forma local
- Contar con cualquier de los navegadores existentes

4. PROCEDIMIENTOS DE DESARROLLO

El desarrollo del chat se fue haciendo por partes. A continuación, se ira explicando cada una de las partes del código del chat para entender su funcionamiento.

4.1 FUNCIONAMIENTO

El chat cuenta con un login para ingresar sus datos, así como su opción para registrarse en caso de que no lo esté. Además de ello, el usuario puede ingresar a la sala que él quiera de las salas disponibles.

Una vez que ingresa, se muestra el chat, con sus respectivos datos correspondientes al usuario loggeado y el espacio donde se muestran los mensajes existentes a la sala a la que entro. De igual manera, se encuentra el espacio para que el usuario puede escribir un nuevo mensaje, así como la opción para que cambie de sala si así lo desea. Aunado a esto, se coloca una opción para cerrar sesión y salir del chat.

DESARROLLO

4.2 DECLARACIÓN DE LIBRERÍAS, CONFIGURACIÓN DEL SERVER, RUTA Y DE LA BASE DE DATOS.

Declaración de las librerías utilizadas.

- Definir como constantes las variables donde hacemos el **require** para utilizar dicha librería. Esto se ve de la siguiente manera:

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

Nota: Para esto, previamente se instalaron, desde la terminal, para ello solo basta colocar en la terminal `npm install` y el nombre de la dependencia que desea instalar, como se muestra en la siguiente imagen:

```
ChatSesionesComplemento> npm install mysql
```

- Declaración de nuestra variable con la que se trabajó el server y lo inicializamos en el puerto deseado. Esto con fin de ahí montar nuestro chat. De la siguiente manera:

```
const port = 3030;

//Declaramos nuestra variable app con la que manejaremos n
uestro server (express)
var app = express();
const nameBot = 'BotMez';

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

- Después de esto se declararon los parámetros necesarios para configurar las sesiones, así como los datos de conexión de la base de datos.

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

Parámetros para configurar las sesiones

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

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

Declaración de la variable para usar socket con el servidor y así conectar las sesiones con el socket. Para hacer uso de ellas entre el servidor y el cliente

Creación de la conexión a la base de datos “chat”

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

Datos de conexión con la base de datos de forma local

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

Especificarle al servidor con lo que va a trabajar

RUTA

Para simplificar el trabajo de las rutas, se hizo uso de una sola ruta estática.

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

Hecho todo esto, se tiene configurado el servidor y listo para usarse.

4.3 LOGIN

Se definió una función con socket.on para recibir los datos que ingresa el usuario, para ello se define una variable para cada uno de los datos.

```
const user = data.user, //Obtenemos el nombre usuario
      pass = data.pass, //Obtenemos la contraseña del usuario
      roomId = data.roomID, //Obtenemos el ID de la sala
      roomName = data.roomName; //Obtenemos el nombre de la sala
```

Después se realizó una validación para verificar si el usuario existe o no, en la base de datos, con la siguiente consulta:

```
"SELECT * FROM users WHERE Username=?", [user]
```

Esta consulta, se hizo mediante una **query** que hace la llamada a la base de datos para poder realizar dicha consulta y a continuación se valida. Si el

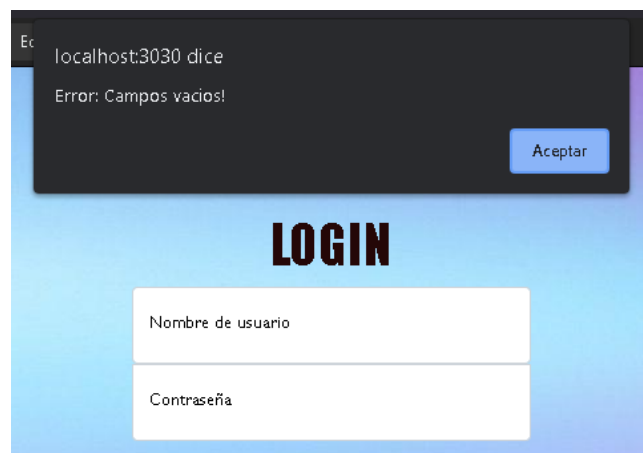
numero de columnas es igual que 0, significa que no existe dicho usuario que intenta ingresar al chat, si no es igual que 0, entonces procede a extraer esos datos de la base de datos y trabajar con ellos.

```
//Revisamos que si este dado de alta en la base de datos, con el user
name
db.query("SELECT * FROM users WHERE Username=?", [user], function(err,
rows, fields){
    if(rows.length == 0){
        socket.emit("error");
    }else{
        const userid = rows[0].id,
        dataUser = rows[0].Username,
        dataPass = rows[0].Password,
        dataCorreo = rows[0].email;
```

Entonces lo que siguió fue hacer una consulta para verificar si la caja donde debe de colocar el usuario su username y contraseña no se dejaron en blanco y después se pregunta si coinciden dichos datos con los de la base de datos.

```
//Validamos. Si la contraseña y usuario son nulos ->
if(dataPass == null || dataUser == null){
    socket.emit("error"); //se manda con un socket.emit un error
}
if(user == dataUser && pass == dataPass){
    console.log("Usuario correcto!"); //si si, ingresa correctamente
```

Código donde se valida que nos campos no estén vacíos y además que coincidan los datos ingresados con los de la base de datos

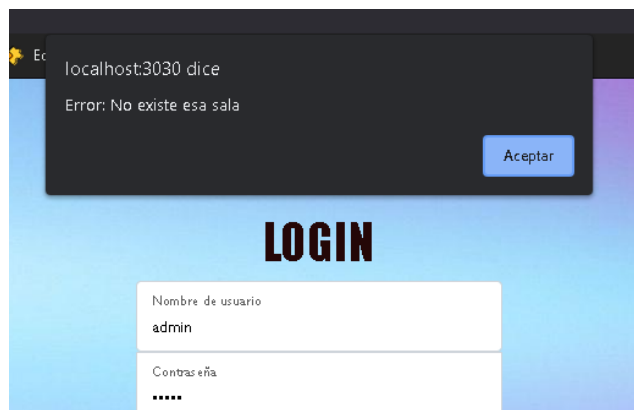
The image shows a web browser window with a dark theme. At the top, there's a notification box with the text "localhost:3030 dice" and "Error: Campos vacíos!". Below this, there's a blue button labeled "Aceptar". The main part of the page has a light blue gradient background. In the center, the word "LOGIN" is written in large, bold, black letters. Below "LOGIN", there are two white input fields. The first field is labeled "Nombre de usuario" and the second field is labeled "Contraseña".

Alerta de que los campos están vacíos

Si todo sale bien, se realiza a continuación una consulta más, para extraer las salas de la base de datos y revisar que el usuario allá ingresado a una base de datos existente. Si es así entonces se realiza un emit a la función `logged_in` (que explico de ella mas tarde), para así, mandar los datos que ingreso el usuario en un json.

```
db.query("SELECT * FROM salas where id= ?", [roomID], function(err,rows,fields){
    if(rows.length == 0)
    {
        socket.emit("errorS");
    }else{
        socket.emit("logged_in", {
            user: user, //usuario
            email: dataCorreo, //correo
            roomID: roomID, //id de la sala
            roomName: roomName}); nombreSala
```

En caso de que el usuario elije una sala no existente, se le muestra una alerta de que esa sala no existe.



Alerta de que esa sala no existe

LOGGED_IN

La función de **logged_in**, se creó en el archivo html para ahí mostrar en un apartado la información con la que ingreso el usuario. Además de hacer un emit a "historial" para que muestre todos los mensajes de la sala una vez que el usuario ingresa.

```
socket.on("logged_in", function(data){
  console.log(data);
  $(".form-signin").hide();
  $("#wrapper").show(); //mostramos el nombre y email del usuario que
  inicio sesión
  $('#usernameTag').text(data.user);
  $('#emailUser').text(data.email);
  $('#roomTag').text(data.roomName); //ademas del nombre de la sala
  socket.emit('historial');
});
```

Código de la función logged_in

```
Session {
  cookie: { path: '/', _expires: null, originalMaxAge: null, httpOnly: true },
  userID: 4,
  Username: 'modelmz',
  correo: 'mmondragon@ucol.mx',
  roomID: '3',
  roomName: ' Probabilidad'
}
```

Datos de la sesión

Dichos datos guardados en el json, posteriormente los mandamos a la sesión, de modo que el usuario podrá iniciar sesión y se guardan en la sesión. Posteriormente colocamos la instrucción de **socket.join** para que el usuario entre a la sala que eligió.

De igual manera se hace un emit a la función que trae todos los mensajes ya registrados en esa sala. Así como el mensaje de nuestro bot, dando la bienvenida.

En caso de que los datos ingresados por el usuario no coincidan, mandamos una alerta con un emit de que son inválidos.

```
//toda esa info la mandamos a la sesión
req.session.userID = userid;
req.session.Username = dataUser;
req.session.correo = dataCorreo;
req.session.roomID = roomID;
req.session.roomName = roomName;
req.session.save(); //guardamos esos datos en la sesión

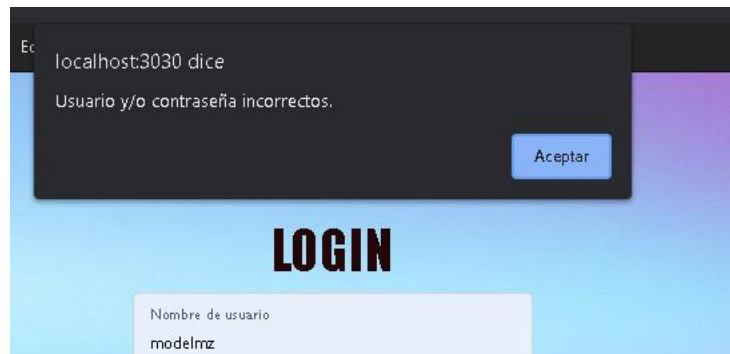
socket.join(req.session.roomName); //entramos a la sala elegida
socket.emit('juntarHisto'); //emit con el historial de la sala
bottxt('entrarSala'); //mensaje del bot de que entro a la sala
//console.log(req.session);
```

```

    }
  });
} else {
  socket.emit("invalido");
}
}
});
});
});

```

Código donde se mandan los datos de la sesión e ingresa a la sala.



Alerta en caso de que ingrese datos incorrectos

BOTON DE LOGIN

Otra función que se encuentra en el html es la del evento del **botón de ingresar** en el login. Al momento de que se presiona dicho botón se hace un emit a la función de login (que ya explicamos) con toda la información recuperada en las cajas de texto.

```

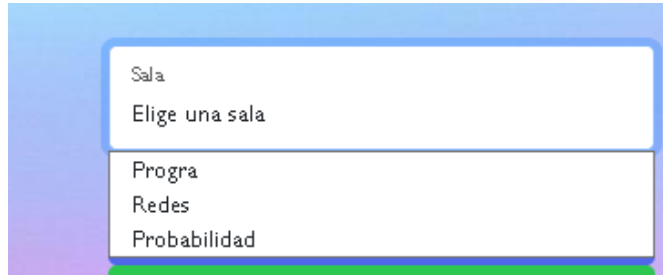
//Función para el login
$("#Login").click(function(){
  socket.emit("login", {
    user: $("#userName").val(), //recibimos el valor para username
    pass: $("#Password").val(), //recibimos el valor para password
    roomID: $("#rooms").val(), //recibimos el valor id de la sala
    roomName: $("#rooms").find('option:selected').text()

    //estos parametros los usamos en la función de login en el index.js
  });
});
});

```

LLENADO DEL SELECT DEL LOGIN

Para poder hacer el llenado del select, para que el usuario pueda elegir la sala que quiera, como se muestra en la siguiente imagen:



Se tuvo que realizar una función llamada “showRooms”, en la cual se creo un array llamado showSalas. Entonces mediante un each se hizo una función para ver cuantas salas existentes hay. Esto se realizo con una consulta, la cual esta en el js donde se hace un emit a showRooms, para que mande la información de las salas.

```
db.query("SELECT * FROM salas", function(err,rows,fields){
  socket.emit("showRooms", rows); //Hacemos el emit de showRooms
});
```

Teniendo ya la información de las salas existentes, se extrajo solamente el id y el nombre de la sala y con eso llenamos el array previamente creado.

```
socket.on("showRooms", function(salas){
  let showSalas = new Array(); //creamos un array donde guardarlas
  $.each(salas, function(id, val){
    var infoR = {id: salas[id].id, nombre_sala: salas[id].nombre_sala}
    showSalas.push(infoR); //hacemos un push al array
  });
```

Como ya se tiene la información contenida en el array, solo es cuestión de colocarla en el select, mandando llamar al array y recorriéndolo con un for, para llenar una variable creada con las opciones para el select. Quedando de la siguiente manera:

```
let infoSalas = '<option hidden selected>Elige una sala</option>';
showSalas.forEach(element => {
  infoSalas+= `<option value = ${element.id}> ${element.nombre_sala}<
/option>`;
});
```

Aquí indicamos con el id del select, que se va a llenar con la información contenida en infoSalas.

```
document.getElementById("rooms").innerHTML = infoSalas;  
document.getElementById("roomCambio").innerHTML = infoSalas;
```

4.4 REGISTRAR USUARIOS

Se cuenta de igual manera con una opción para registrar a usuarios que no estén dados de alta en la base de datos. Para esta función de igual forma, se creó con el **socket.on**, llamada **addUser**.

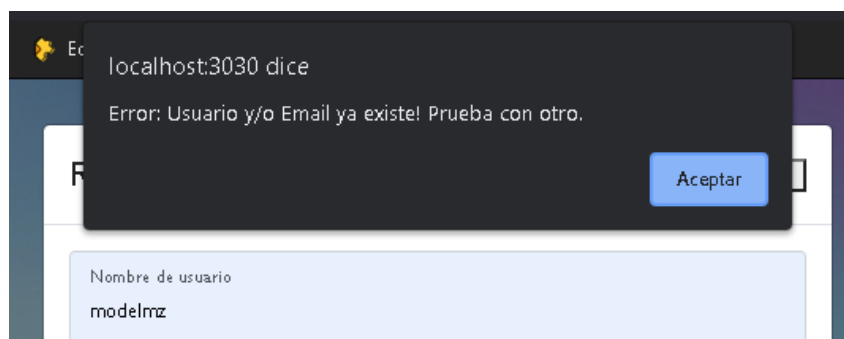
```
socket.on('addUser', function(data){
```

Dentro de ella, de igual forma que en el login, se definió una variable para cada uno de los datos que pertenecen al usuario para poder registrarse, es decir, su nombre de usuario, contraseña y correo.

Después se hizo una validación, para revisar que todos los campos no estén en blanco. Ya que se comprueba esto, se hace una consulta para revisar si los datos ingresados no existen ya en la base de datos.

La validación dice que si rows es igual a 0, significa que no hay datos duplicados y entonces si se puede agregar y para ella hacemos la consulta para agregar ese usuario en la base de datos.

En caso de que rows, sea mayor que 0, significa que los datos del correo o del usuario ya son existentes y se le mandara una alerta, como la siguiente:



Alerta de usuario o correo ya existente en la base de datos

```

const user = data.user,
pass = data.pass,
email = data.email;

//si el user, pass y email, son diferentes a espacios vacíos
if(user != "" && pass != "" && email != ""){
    console.log("Registrando el usuario: " + user);

    db.query('SELECT * FROM users where Username = ? or email = ?', [user,
    email], function(err, rows, res){
        if(rows.length == 0){

            db.query("INSERT INTO users(`Username`, `Password`, `email`) VALUES(
            ?, ?, ?)", [user, pass, email], function(err, rows, result){
                if(!err){
                    throw err; //si algo sale mal, manda un error
                }else{
                    console.log('Usuario ' + user + " se dio de alta correctamente
                    !.");
                    socket.emit('UsuarioOK');
                }
            });
        }else{
            socket.emit('errorRegis');
        }
    });
}

}else{ //alerta de que dejo campos vacios
    socket.emit('vacio');
}
});

```

Código para la función del registro de usuarios

BOTON PARA REGISTRAR AL USUARIO

Para que el registro sea existo, debemos de agregarle el evento al botón de registrar con el click. El código de abajo muestra que al hacer el click, se hará un emit con la información recabada en las cajas de texto a la función de **addUser** para que inserte ese nuevo usuario.

```

$("#sendResgistro").click(function(){
    socket.emit("addUser", {
        user: $("#userNameR").val(), //recibimos el username
    });
});

```



```

    pass: $("#PasswordR").val(), //con su contraseña
    email: $("#correo").val(), //y su correo

    //estos parametros los usamos en la función de addUser en el index.
js
  });
});

```

Código para el evento del botón de registro

4.5 CAMBIAR DE SALA

La siguiente función sirve para que el usuario pueda cambiar de sala, aunque ya allá entrado al chat. Esto con el fin de hacer más interactivo nuestro chat.

Entonces primero se crea la función, la cual se llama cambiaSala. Dentro de ella se crean las variables las variables para traer el id de la sala como el nombre de esta.

```

socket.on('cambiarSala', function(data){
    const IDroom = data.IDroom, //Obtenemos el id de la sala
    nameRoom = data.nameRoom; //Obtenemos el nombre de la sala

```

Entonces como el usuario va a hacer un cambio, se debe de salir de la sala en la que esta actualmente, esto se hace con la siguiente línea de código:

```

socket.leave(req.session.roomName); //salimos de la sala

```

Hecho esto, ahora se igualan los nuevos datos de la sala, con los datos de la sesión, para que estos cambian y pueda hacerse el cambio de sala

```

req.session.roomID = IDroom;
req.session.roomName = nameRoom;

```

Finalmente, solo se hace un join, con la nueva sala, así como el mensaje del bot de que cambio de sala.

```

socket.join(req.session.roomName); //entramos a la nueva sala
bottxt('cambiarSala');//mensaje del bot de que cambio de sala

```

Para aplicar un cambio de sala, ya una vez que el usuario está dentro del chat, debemos de hacerlo con un .change al select y obtener los nuevos valores de la sala.

CAMBIO DE SALA EN EL SELECT Y EN EL ROOMTAG

Ya que se hace eso, también se cambia el nombre de la sala, en el apartado donde se ven los datos del usuario y además vaciamos el chat, ya que ahora se mostrarán los nuevos mensajes de la nueva sala, haciendo el emit a la función de **cambiar sala**. La última instrucción para que mande el historial de los mensajes de la nueva sala.

```
//Funcion para hacer el cambio de sala
$('#roomCambio').change(function(){
    var roomID = $(this).val(); //recibimos el valor del id
    var roomName = $(this).find('option:selected').text();

    $('#roomTag').text(roomName); //hacemos el cambio en el roomTag
    $('#chatbox').empty();

    socket.emit('cambiarSala',
        IDroom: roomID,
        nameRoom: roomName
    });

    socket.emit('historial');
});
```

Codigo del roomsCambio del select

También se agregó una función para salir del chat al momento que desee el usuario. Simplemente se hace un socket.leave, para salir de la sala y se destruye la sesión.

```
socket.on('salir', function(request, response){
    socket.leave(req.session.roomName);
    req.session.destroy();
    //si presionamos el boton de salir, se destruye la sesión con la que se
    e habia ingresado
});
```

4.6 MENSAJES

Se creó la función **msjNuevo**, donde dentro de ella, se realizó la consulta para insertar un mensaje, una vez que el usuario presionó el botón de “enviar mensaje”. Dentro de esa función, se hizo una consulta para agregar ese mensaje a la base de datos

```
db.query("INSERT INTO mensajes(`mensaje`, `user_id`, `sala_id`, `fecha`)  
VALUES(?, ?, ?, CURDATE())", [data, req.session.userID, req.session  
.roomID],
```

Y después de esto, si en la validación todo salió bien, se hace un `broadcast.to` del mensaje, el dato de la sesión en la sala que esta, para enviarlo solo a esa sala y no a todas, con la información de quien lo mando y el contenido del mensaje. Después se hace otro `emit` para mandar el mensaje y se vea en el chat inmediatamente.

```
socket.broadcast.to(req.session.roomName).emit('mensaje',{  
    usuario: req.session.Username, //con el nombre de usuario de la  
sesion  
    mensaje: data //y el mensake  
});  
  
//de igual manera hacemos un emit del mensaje (es decir q lo envíe y s  
e vea)  
socket.emit('mensaje', {  
    usuario: req.session.Username, //con el nombre de usuario  
    mensaje: data //y con el mensaje  
});
```

Código del `emit` del mensaje a la sala en la que se encuentra

BOTON ENVIAR MENSAJE

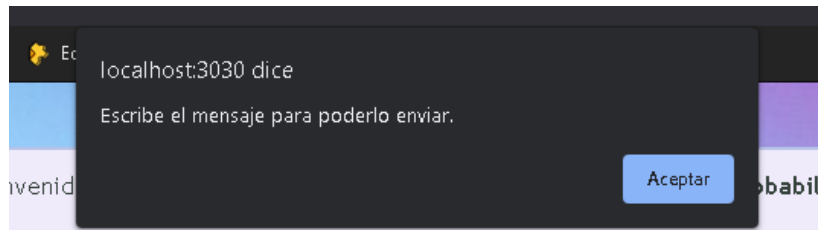
En el html, se hizo función para el evento del botón que envía el mensaje. Se hizo una validación para en el caso de que no se halla escrito nada, te mando un error de que no has escrito dicho mensaje. Si no es así, mandamos el valor del mensaje obtenido de la caja de texto y hacemos un `emit` a la función de `mjsNuevo` para que agregue ese nuevo mensaje a la base de datos.

```
//Función para mandar el mensaje  
$('#enviarMensaje').click(function(){  
    if($("#mensaje").val().length <= 0){
```

```

        alert("Escribe el mensaje para poderlo enviar.");
    }else{ //si no, es que si esta escrito
        var mensaje = $('#mensaje').val()
        socket.emit('mjsNuevo', mensaje);
    } // Enviamos el nuevo mensaje a la función de mjsNuevo en el index.js
    }
});

```



Bienvenido a la sala Probabilidad

Alerta de que debe de escribir el mensaje

La siguiente función es la que encargada de recibir el nuevo mensaje con el socket.on y mostrarlo en el chatbox, con el nombre de usuario y su respectivo mensaje.

```

//Función que tiene de respuesta el nuevo mensaje
socket.on('mensaje', function(data){
    var nuevoMensaje = '<b>' + data.usuario + ' dice: </b>' + data.mensaje;
    $('#chatbox').append(nuevoMensaje + '<br>');
    $('#mensaje').val("");
});

```

4.7 HISTORIAL DE MENSAJES

Para interactuar mas los usuarios entre las salas, se implementó el historial de mensajes, para que cada vez que un usuario inicia sesión en cualquier sala, aparezca el historial de mensajes de ella.

Para ello, se creó una función llamada "historial", donde se hizo una consulta que trae de vuelta los mensajes y el nombre de quien lo mando de una sala en específico. Una vez que se hace dicha consulta, se hace un emit a "juntarhisto", con la información recabada en rows.

```

socket.on('historial', function(data){

```

```

    console.log('Busqueda de historial en la sala: ' + req.session.roomName);

    //consulta que devuelve el nombre de usuario, la sala y el mensaje (para ver los mensajes de esa sala)
    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('juntarHisto', rows); //emit a la función que une todos los mensajes
    });
});

```

JUNTAR HISTORIAL

Esta función básicamente contiene el formato de como se muestran los mensajes para crear así el historial. Solamente se tiene que recorrer la información y guardarla en una variable con los datos que quieres mostrar y los mandas al chatbox. Esto en código se ve de la siguiente manera:

```

socket.on('juntarHisto', function(data){
    var historial=''; //se crea una variable para ahí guardarlos
    $.each(data, function(id, val){ //hacemos un foreach para extraer todos los mensajes
        historial += `<p><b>` + data[id].Username + ` dijo : </b>` + data[id].mensaje + `</p>`;
        //mandamos el nombre de usuario y el mensaje
    });
});

```

Mostrarlo en el chat

```

$('#chatbox').append(historial);

```

Bienvenido, **modelmz**, con correo: **mmondragon@ucol.mx** a la sala **Redes**

[Salir del chat](#)

BotMez dice: Bienvenido a la sala Redes

modelmz dijo : Holi, esta es una pruebita

modelmz dijo : Holi, esta es una pruebita

modelmz dijo : solo sabes decir eso?

modelmz dijo : HOLA

admin dijo : holi

modelmz dijo : holi

admin dijo : eyy volvi

modelmz dijo : como t fue

modelmz dijo : Ey, ya se ve mas bonito!!

modelmz dijo : holi

modelmz dijo : holi

Historial de mensajes de la sala Redes

Esta función, se hace un emit de ella al momento en el que el usuario inicia sesión para que así se le muestre el historial al momento de que ingresa.

```
req.session.roomID = roomID;
req.session.roomName = roomName;
req.session.save(); //guardamos esos datos
socket.join(req.session.roomName); //entrar al chat
socket.emit('juntarHisto'); //emit con el historial
bottxt('entrarSala'); //mensaje del bot de bienvenida
```

De igual forma en el evento de roomsCambio, y en el logged_in se manda la información de historial, para que se haga la consulta y haga el emit de la información a juntarHisto.

```
//Funcion para hacer el cambio de sala
$('#roomCambio').change(function(){
    var roomID = $(this).val();
    var roomName = $(this).find('option:selected').text();

    $('#roomTag').text(roomName);
    $('#chatbox').empty(); //vaciar el chat

    socket.emit('cambiarSala', {
        IDroom: roomID,
        nameRoom: roomName
    });

    //comprobar que si se hizo el cambio
    //console.log('Cambio seleccionado');
    socket.emit('historial');
});
```

```
//Funcion de la creación de la sesión
socket.on("logged_in", function(data){
    console.log(data);
    $(".form-signin").hide();
    $("#wrapper").show(); //mostar el wrapper
    $('#usernameTag').text(data.username);
    $('#emailUser').text(data.email);
    $('#roomTag').text(data.roomName);
    socket.emit('historial'); //emitar el historial
});
```

Emit de la función historial en logged_in y roomsCambio

4.8 BOT

De igual manera, se agrego un bot, para que el usuario reciba mensajes de los cambios que realiza en el chat, por ejemplo, cuando ingreso o cuando cambio de sala.

BotMez dice: Bienvenido a la sala Redes

BotMez dice: Haz cambiado de sala. Tu nueva sala es: Probabilidad

Mensajes del Bot

Esto, se hizo de la siguiente manera. Primero se creó una función especialmente para el bot.

```
function bottxt(data){
```

Donde dentro de ella, se definieron dos mensajes, el de entrar sala y de cambio de sala, y hacemos comparaciones, si el mensaje es **entrarSala**, se hace un emit de los mensajes, para que lo muestre. A continuación, se muestra el código:

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

Y ¿cuándo se le da la bienvenida al usuario? Cuando ingresa, es por ello, que de igual manera, cuando el usuario entra (cuando se hace el join), se manda llamada a la función del bot con el mensaje de Bienvenida.

```
req.session.roomID = roomId;
req.session.roomName = roomName;
req.session.save(); //guardamos esos datos en la sesión
socket.join(req.session.roomName); //entramos a la sala elegida
socket.emit('juntarHisto'); //emit con el historial de la sala
bottxt('entrarSala'); //mensaje del bot de que entro a la sala
```

Llamada a la función del bot para que de la bienvenida

De igual forma con el mensaje de cambiarSala, se hace una comparación, si es igual a ese mensaje, se hace el emit al mensaje para que lo muestre y hacemos una llamada a la función cuando se hace el cambio.

```
socket.on('cambiarSala', function(){
  const IDroom = data.IDroom,
  nameRoom = data.nameRoom; //

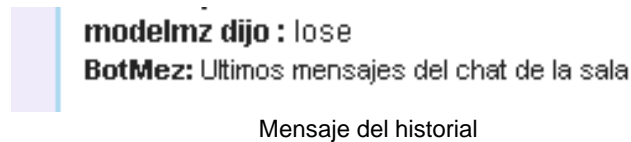
  socket.leave(req.session.roomName);

  //igualamos los nuevos datos
  req.session.roomID = IDroom;
  req.session.roomName = nameRoom;

  socket.join(req.session.roomName);
  bottxt('cambiarSala'); //mensaje del bot de que cambio de sala
});
```

Llamada a la función del bot para que diga que cambio de sala

Así mismo en la función de juntarHisto, el bot manda un mensaje diciendo que esos son los últimos mensajes del chat.



4.9 CREACIÓN DE LA BASE DE DATOS

Para el chat se necesito de la creación de una base de datos para administrarlo. Para esto, se creó una base de datos llamads “chat”, con las siguientes tablas:

- mensajes
- usuarios
- salas

Para la creación de todo usamos las siguientes consultas:

a) Creación de la base de datos y seleccionarla

```
create database chat;  
use chat;
```

b) Tabla de mensajes

```
DROP TABLE IF EXISTS `mensajes`;  
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;
```

c) Tabla de salas

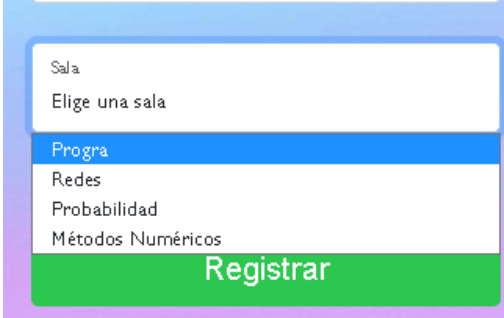
```
DROP TABLE IF EXISTS `salas`;  
CREATE TABLE `salas` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `nombre_sala` varchar(30) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,  
  `fecha_creación` date NOT NULL,  
  PRIMARY KEY (`id`) USING BTREE  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC;
```


d) Tabla de usuarios

```
DROP TABLE IF EXISTS `users`;  
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;
```

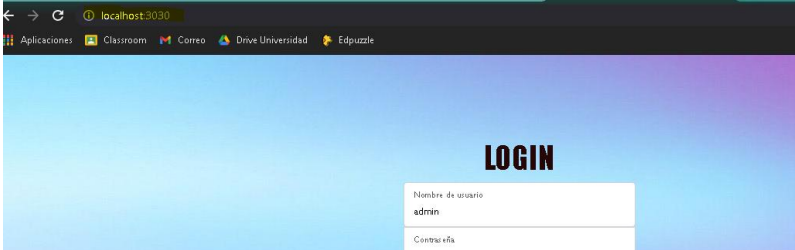
e) Ya que tenemos todo lo demás, insertamos a la tabla de sala 4 salas, para que los usuarios tengan variedad de salas a las que ingresar

```
INSERT INTO salas (nombre_sala, fecha_creación) VALUES ('Progra', current_date());  
INSERT INTO salas (nombre_sala, fecha_creación) VALUES ('Redes', current_date());  
INSERT INTO salas (nombre_sala, fecha_creación) VALUES ('Probabilidad', current_date());  
INSERT INTO salas (nombre_sala, fecha_creación) VALUES ('Métodos Numéricos', current_date());
```



Salas en el select

Cabe mencionar que el proyecto chat, se trabajo desde una base de datos de forma local, es por ello, que para ingresar a él, es con localhost [el puerto seleccionado], en este caso se usa, el puerto 3030.



Servidor puesto en marcha en el puerto 3030


5. DISEÑO Y RESULTADOS

Se uso hizo de uso del Bootstrap, además de estilos css para darle estilo y color a nuestro chat, mediante los ids colocados para cada componente o la creación de clases. Se aplicaron estos estilos para los:

- Títulos
- Fondos
- Inputs
- Selects
- Botones
- Tipo de letra
- Colores, etc.

A continuación de muestran imágenes del resultado final de cada parte del código.

LOGIN



LOGIN

Nombre de usuario

Contraseña

Sala
Elige una sala

Ingresar

Registrar

© 2021

Pagina principal. Con los campos para que usuario inicie sesión son su nombre de usuario y contraseña. Además de contar con el select para que el usuario elija la sala que desee.

OPCION DE REGISTRO

Registrar

Nombre de usuario

Contraseña

Correo

Cerrar Registrar!

Ingresar

Registrar

Apartado para hacer el registro de usuarios. Con su nombre de usuario, contraseña y correo.

CHAT AL INGRESAR

Bienvenido, **modelmz**, con correo: **mmondragon@ucol.mx** a la sala **Métodos Numéricos** [Salir del chat](#)

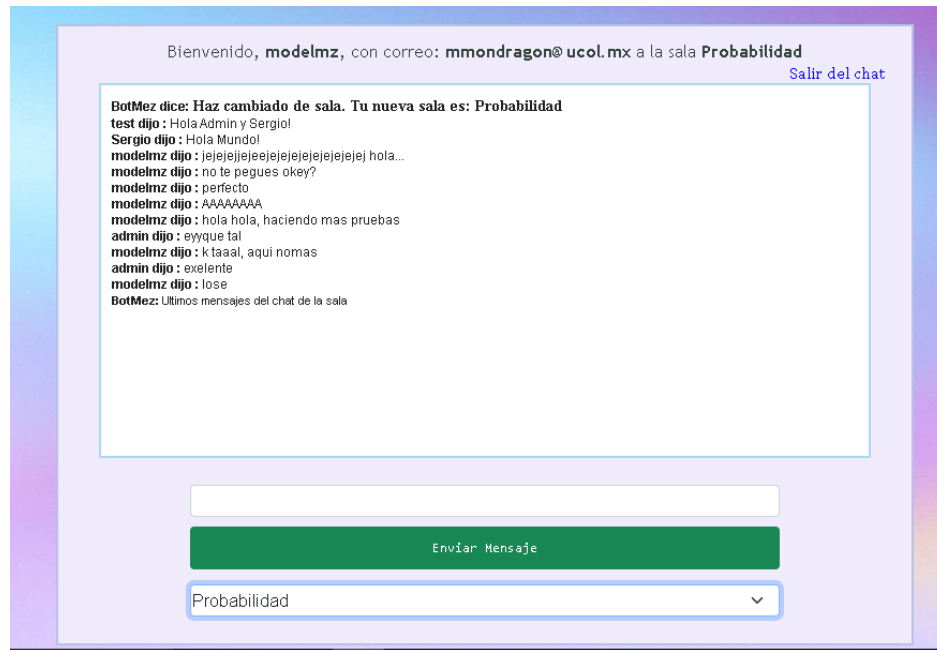
BotMez dice: Bienvenido a la sala Métodos Numéricos
duque dijo : hola, soy nuevo aqui
modelmz dijo : eyy
modelmz dijo : yo tambien soy nueva
duque dijo : vaya, me siento mas tranquilo
duque dijo : gracias humana
modelmz dijo : de nada perruno
admin dijo : holaaa, que tal estan
duque dijo : hola, muy bien y tu?
duque dijo : eres el admin?
admin dijo : jejeje parece que si verdad
BotMez: Ultimos mensajes del chat de la sala

Enviar Mensaje

Elige una sala

Aquí claramente se puede observar, el historial de mensajes de la sala de Métodos Numéricos, con los datos de los usuarios que enviaron los mensajes, así como el mensaje de bienvenida de nuestro bot. Además de que en la parte superior se muestran los datos de la persona que está en sesión.

CHAT AL CAMBIAR DE SALA



Como podemos observar, cambio el dato de la sala, en la parte superior, así como el mensaje del bot y de igual trae el historial de los mensajes de la nueva sala.

GLOSARIO

- **Inputs:** Conjunto de datos que se introducen en un sistema o un programa informáticos.
- **Bot:** Efectúa automáticamente tareas reiterativas mediante Internet a través de una cadena de comandos o funciones autónomas previas para asignar un rol establecido.
- **Bootstrap:** Biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web.
- **Librería:** Conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
- **Query:** Pregunta o consulta. Petición precisa para obtener información en una base de datos o sistema de información.
- **Middleware:** Se encarga de las tareas de gestión de datos, servicios de aplicaciones, mensajería, autenticación y gestión de API.
- **socket.emit:** Emitir un evento de su cliente.
- **Select:** Control que muestra un menú de opciones.

CONCLUSION

Como conclusión, después desarrollar e implementar un chat con las diferentes librerías utilizadas, se puede decir que saber y conocer el manejo de diferentes tipos de librerías ayudan bastante para el desarrollo de aplicaciones web, sobre todo por que facilitan la implementación del código.

Anteriormente había trabajado con JavaScript y con la librería de express y MySQL, pero no con socket.io, ni con express-session y cookie-parser. Es muy interesante ver como es que trabajan este tipo de librerías en conjunto para hacer funcionar todo un pequeño sistema o incluso grandes sistemas. En este caso solo fue un chat, son su respectivo login, incluimos el manejo de salas, así como el manejo de una base de datos, sin ella el chat no tendría sentido ya que no almacenaría nada.

Asimismo, fue una experiencia divertida, el estar probando y experimentando con el código, viendo cómo se comportaba ante diferentes situaciones, como buscando posibles bugs que pudiera tener y viendo que mas cosas se le podrían implementar para mejorarlo, así como realizar las validaciones necesarias para que todo funcionara correctamente.

Después de manejar y desarrollar este sistema, espero poder aplicarlo en proyectos a futuro y claro mejorar esta versión, para implementarle las nuevas tecnologías que puedan ir surgiendo.