

**FACULTAD DE TELEMÁTICA**

**ING. EN TECNOLOGÍAS DE INTERNET**

Interfaz de usuario gráfica

Descripción generada automáticamente**“CHAT CON SESIONES Y SALAS”**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

PROGRAMACIÓN DISTRIBUÍDA

RAMÍREZ GARCÍA CARLOS ANTONIO

PROF. MONTAÑO ARAUJO SERGIO ADRIÁN

4°C

*25/JUNIO/2021*

**ÍNDICE**

[**INTRODUCCIÓN** 4](#_Toc75382297)

[**INSTALACIÓN DE DEPENDENCIAS Y LIBRERÍAS** 5](#_Toc75382298)

[**SERVIDOR** 5](#_Toc75382299)

[DECLARAR DEPENDENCIAS 5](#_Toc75382300)

[ARRANCAR SERVIDOR 5](#_Toc75382301)

[CONFIGURACIÓN DE SOCKET Y EXPRESS 6](#_Toc75382302)

[CONEXIÓN DE BASE DE DATOS 6](#_Toc75382303)

[**FUNCIONES DE SOCKET** 7](#_Toc75382304)

[MOSTRAR SALAS 7](#_Toc75382305)

[VERIFICAR ESTADO DE LA SESION 7](#_Toc75382306)

[FUNCIÓN DE LOGIN 8](#_Toc75382307)

[FUNCION ADDUSER 9](#_Toc75382308)

[FUNCION MENSAJE NUEVO 10](#_Toc75382309)

[FUNCION SALIR 10](#_Toc75382310)

[FUNCIÓN CAMBIO DE SALA 11](#_Toc75382311)

[FUNCIÓN HISTORIAL 11](#_Toc75382312)

[FUNCIÓN NOTIFICACIONES 12](#_Toc75382313)

[**VISTAS HTML** 12](#_Toc75382314)

[VISTA LOGIN 12](#_Toc75382315)

[VISTA DEL CHAT 13](#_Toc75382316)

[VISTA DEL REGISTRO 13](#_Toc75382317)

[**HEAD HTML** 14](#_Toc75382318)

[**ESTILOS** 15](#_Toc75382319)

[**SCRIPTS LIBRERIAS** 15](#_Toc75382320)

[**SCRIPT JAVASCRIPT** 16](#_Toc75382321)

[CONEXIÓN SOCKET 16](#_Toc75382322)

[MOSTRAR SALAS 16](#_Toc75382323)

[BOTÓN LOGIN 17](#_Toc75382324)

[BOTÓN REGISTRAR 17](#_Toc75382325)

[CERRAR SESIÓN 17](#_Toc75382326)

[ENVIAR MENSAJE 18](#_Toc75382327)

[FUNCIÓN LOGGED\_IN 18](#_Toc75382328)

[MOSTRAR HISTORIAL DE MENSAJES 18](#_Toc75382329)

[CONCATENAR NUEVO MENSAJE 19](#_Toc75382330)

[CAMBIO DE SALA 19](#_Toc75382331)

[**ERRORES Y NOTIFICACIONES** 20](#_Toc75382332)

[USUARIO INVÁLIDO 20](#_Toc75382333)

[ERROR USERNAME Y PASSWORD NULL 20](#_Toc75382334)

[ERROR DATOS VACIOS 20](#_Toc75382335)

[REGISTRO CORRECTO 21](#_Toc75382336)

[ERROR USUARIO NO REGISTRADO 21](#_Toc75382337)

[ERROR SALA INEXISTENTE 21](#_Toc75382338)

[ERROR USUARIO EXISTENTE 21](#_Toc75382339)

[NOTIFICACIONES DEL BOTCHAT 21](#_Toc75382340)

[**BASE DE DATOS** 22](#_Toc75382341)

[TABLA USERS 22](#_Toc75382342)

[TABLA SALAS 22](#_Toc75382343)

[TABLA MENSAJES 22](#_Toc75382344)

[**GLOSARIO** 24](#_Toc75382345)

[**CONCLUSION** 25](#_Toc75382346)

# **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.

Texto

Descripción generada automáticamente 

## 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.

Texto

Descripción generada automáticamente

## CONFIGURACIÓN DE SOCKET Y EXPRESS

Declaramos la variable “*io*” para crear la conexión con socket.io.



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

Texto

Descripción generada automáticamente

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

Texto

Descripción generada automáticamente

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



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



## 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.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# **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.



## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 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:

* 1. Que el username no exista en algún otro usuario.
  2. Que el email no exista en algún otro usuario.

Texto

Descripción generada automáticamenteSi 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.

## 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’)*.**

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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***.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

# **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.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## VISTA DEL CHAT

Texto

Descripción generada automáticamenteMostramos 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.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## 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.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# **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.

Texto

Descripción generada automáticamente

# **ESTILOS**

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

Texto

Descripción generada automáticamente

# **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.

Texto

Descripción generada automáticamente

# **SCRIPT JAVASCRIPT**

## CONEXIÓN SOCKET

Declaramos la variable socket para crear la conexión con 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

También se resetean los inputs.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

# **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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## ERROR DATOS VACIOS

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

Texto

Descripción generada automáticamente

## REGISTRO CORRECTO

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

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

## 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.

Texto

Descripción generada automáticamente

# **BASE DE DATOS**

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



## TABLA USERS

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

Texto

Descripción generada automáticamente con confianza media

## TABLA SALAS

Se creó para almacenar las salas de chat disponibles.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## TABLA MENSAJES

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

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente con confianza media

# **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.