INSTITUTO TECNOLÓGICO DE CHIHUAHUA II

PRESENTA:

|  |  |  |
| --- | --- | --- |
| Barrio Pacheco Mónica Liliana | Méndez Martínez Moisés R. | Hinojos Quintana Rodrigo Daniel |
| Varela González Nohemí Berenice | Rivera Galaviz Yaritza | Flores Portillo César |
| Valdez Diana Rubí | Ximello Ríos Josué | Torres Báez Jovana Jazmín |
| Montoya Gámez Karen | Trejo Ramos Adrián Enrique |  |

CARRERA:

**INGENIERÍA INFORMÁTICA**

CHAT APP

MATERIA:

**DESARROLLO DE APLICACIONES MOVILES**

PERIODO ESCOLAR:

**AGOSTO – DICIEMBRE 2017**

Chihuahua, Chih. A 16 de octubre del 2017

Contenido

[CHAT APP 1](#_Toc495874075)

[¿Por qué usar NodeJS y Socket.io? 1](#_Toc495874076)

[Ejemplo del funcionamiento del chat. 2](#_Toc495874077)

[Estructura. 3](#_Toc495874078)

[Instalación de dependencias. 4](#_Toc495874079)

[Configuración del lado del cliente. 5](#_Toc495874080)

[Código para ventana de login. 5](#_Toc495874081)

[Componentes para ventana del chat y mensajes. 5](#_Toc495874082)

[Código para proceso de registro de usuario. 6](#_Toc495874083)

[Modelo de la base de datos. 7](#_Toc495874084)

[Configuración del lado del servidor. 7](#_Toc495874085)

[Configuración de la base de datos. 7](#_Toc495874086)

[Envío de mensajes y guardado en BD. 8](#_Toc495874087)

[Primera prueba. 10](#_Toc495874088)

[Funcionalidades extra. 13](#_Toc495874089)

[Zumbido. 13](#_Toc495874090)

[Estado. 15](#_Toc495874091)

[Emojis. 17](#_Toc495874092)

[Vista del chat. 19](#_Toc495874093)

# CHAT APP

Documentación acerca de la creación de un chat con NodeJS, Express, Socket.io y mongodb.

Este chat cuenta con las siguientes funciones:

* Login.
* Manejo de usuarios (Muestra los que estan en línea en ese momento).
* Vista del chat.
* Envío de mensajes en chat privado.
* Envío de mensajes en chat público.
* Envío de zumbido en chat privado.
* Envío de emojis.
* Estado de los usuarios (conectado, desconectado).

## ¿Por qué usar NodeJS y Socket.io?

NodeJS es bueno para aplicaciones real time, en las cuales se necesita mantener una conexión persistente entre el browser y el servidor, como por ejemplo en un chat. Aparte permite usar JavaScript tanto del lado del cliente como del servidor y aparte cuenta con una buena gestión de librerías NPM. A parte es capaz de manejar un gran número de conexiones simultáneas con alto rendimiento, lo que equivale a una alta escalabilidad.

Socket.io es una librería JavaScript permite una comunicación bidireccional en tiempo real entre cliente y servidor.

### Ejemplo del funcionamiento del chat.

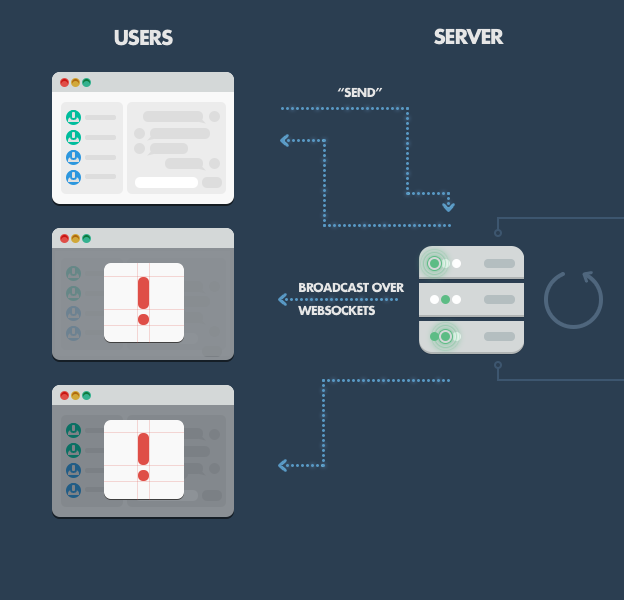
En la figura 1 se visualiza un esquema que representa como es la comunicación con el chat.

Figura Envío de mensajes en el chat.

## Estructura.

En la figura 2 se puede ver la estructura de los archivos que va a tener este proyecto.

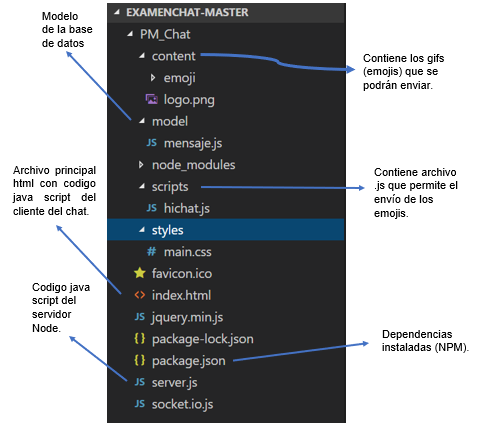


Figura Estructura del proyecto del chat.

## Instalación de dependencias.

Primero hay que instalar las dependencias a utilizar que son:

* Express

Usar JavaScript del lado del servidor.

* Node.
* Mongoose: Para almacenar los mensajes.
* Socket.io: Comunicación bidireccional en tiempo real entre cliente y servidor
* Body- parser: utilizado para el parseo.

El package.json (figura 3) contiene la configuración de la aplicación, en él se encuentran los módulos instalados.

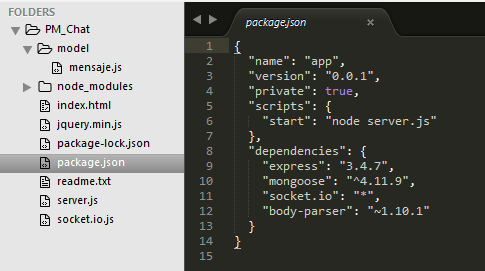


Figura Dependencias instaladas.

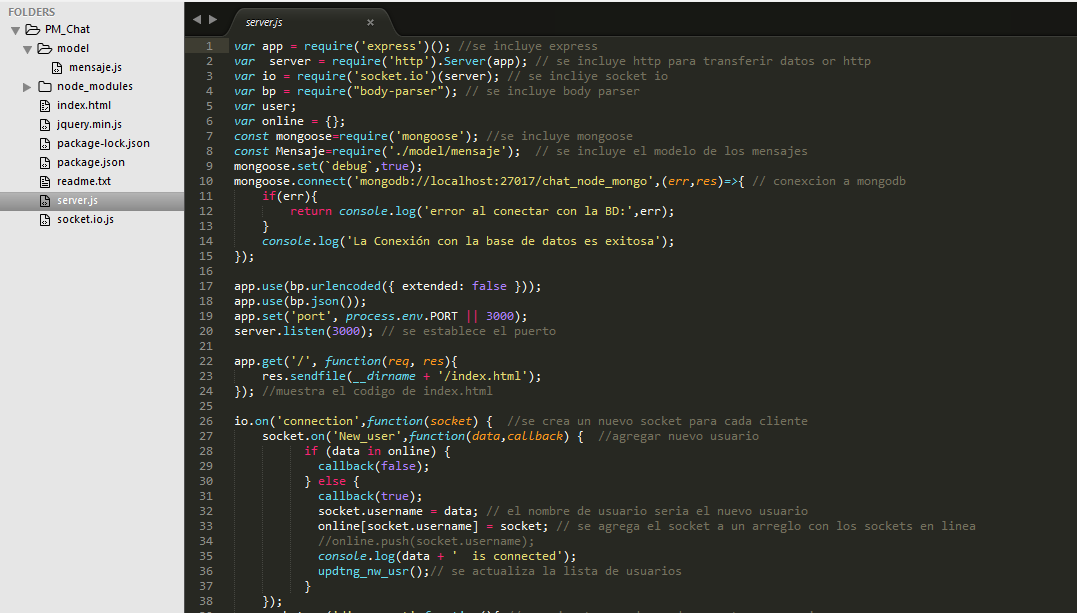
Como se puede ver en la figura 3 se menciona que el archivo principal es server.js, por lo cual es ahí donde se va a configurar la aplicación. Entonces dentro de este archivo (server.js) se andan llamar los módulos para poder hacer uso de ellos (figura 4).

Figura Llamado de módulos.

## Configuración del lado del cliente.

### Código para ventana de login.

En la figura 5 se agrega un input de tipo texto y un botón los cuales son los primeros en mostrarse cuando un usuario va a entrar al chat, es el login.

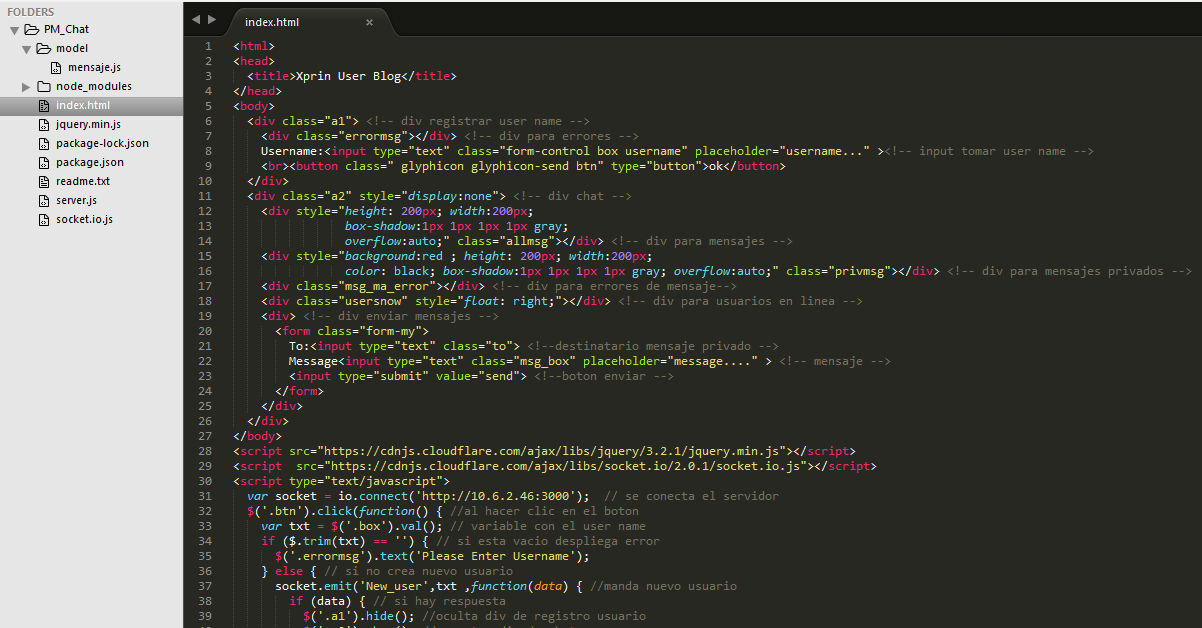


Figura Código para ventana de login para el chat.

### Componentes para ventana del chat y mensajes.

En la figura 6 se establecen los componentes que tendrá la ventana del chat grupal y privado, como se puede ver tiene dos inputs de tipo texto que son para especificar el destinatario y el otro es para escribir el mensaje a enviar y un botón para enviar el mensaje.

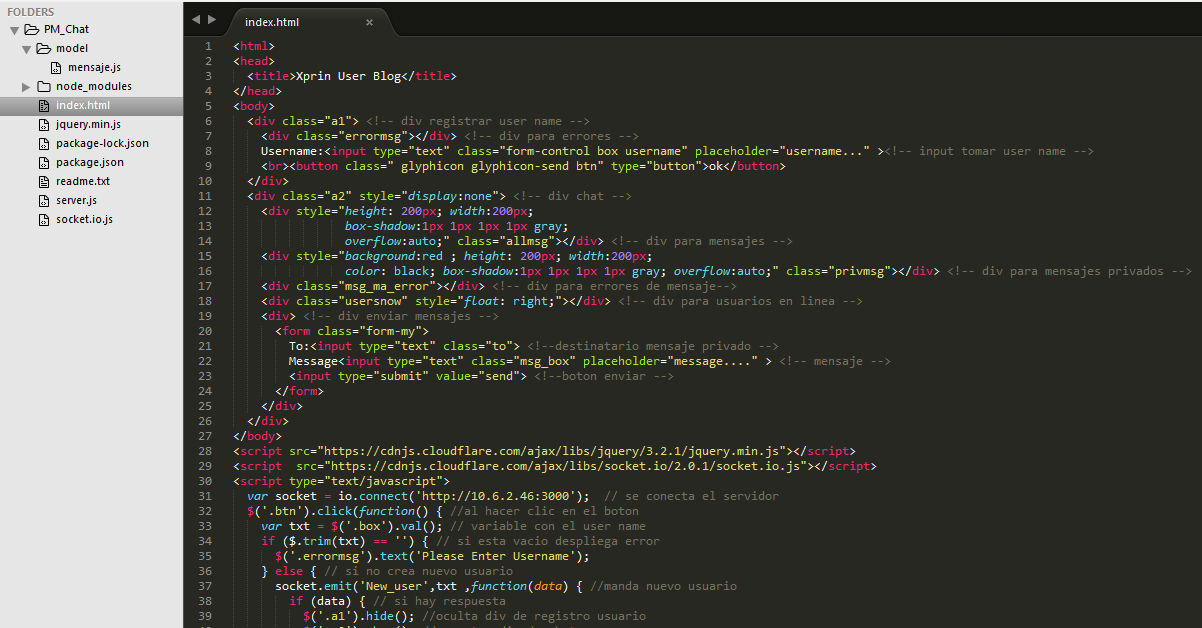


Figura Ventana del chat.

### Código para proceso de registro de usuario.

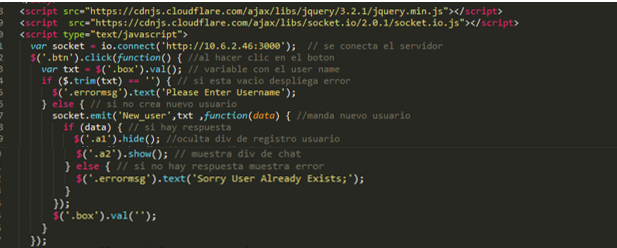
Para conectarse al chat hay que establecer conexión con el servidor en este caso 10.6.2.46 que va a estar escuchando por el puerto 3000 que se configura del lado del servido (server.js) al abrir el chat se mostraran los componentes mencionados en la figura 5 y hay que loguearse si se deja vacío el campo de nombre y se da ok mandara un mensaje de que tiene que poner nombre de usuario (figura 7), en cambio sí llena el campo muestra la ventana de chat.

Figura Código para registro usuario.

Al acceder al chat se actualiza la lista de usuarios disponibles, en el código de la figura 8 se crean eventos para el envío de mensajes públicos, privados y actualización de estado (mientras está escribiendo).

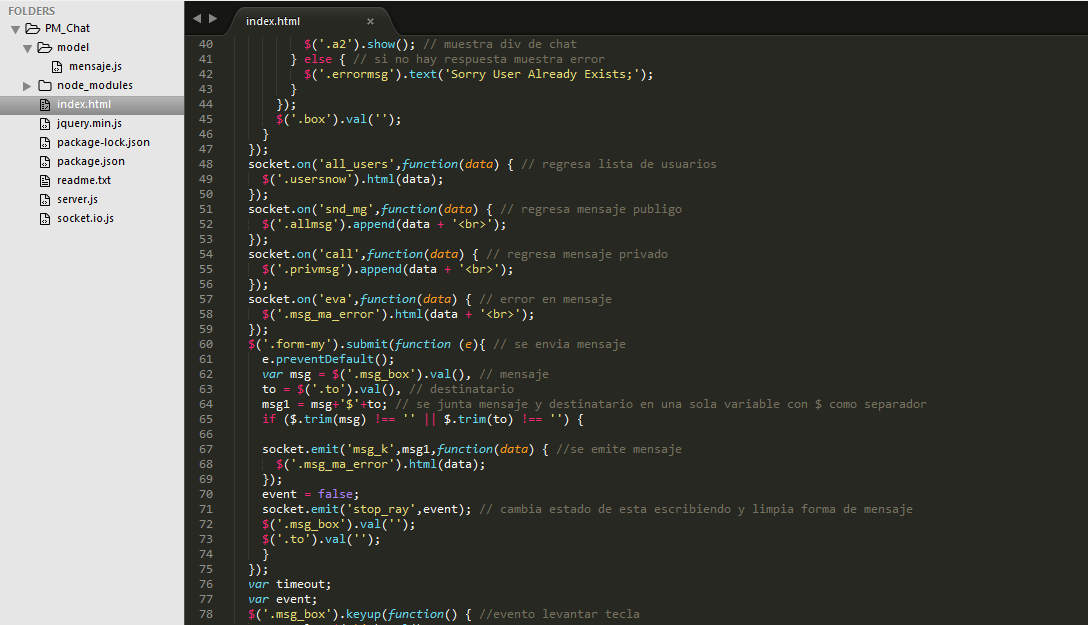


Figura Eventos.

## Modelo de la base de datos.

Dentro de modelos se crea un archivo mensajes.js en el cual se configura los campos que guardar la base de datos (mongoose) y que tipo de dato es (figura 9).

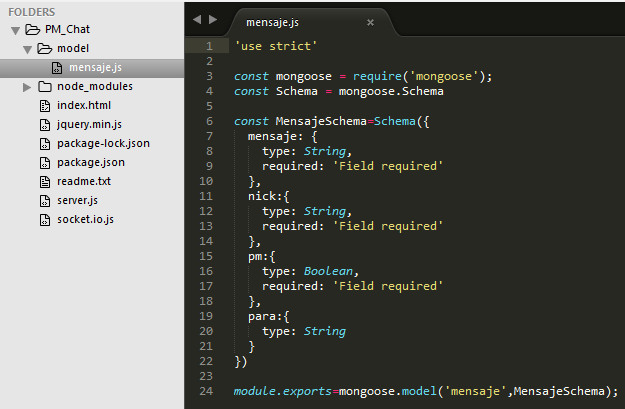


Figura Modelo de la base de datos.

## Configuración del lado del servidor.

### Configuración de la base de datos.

Ahora dentro de la server.js se va a configurar la base de datos MongoDB y el puerto por el cual va a estar escuchando (figura 10).

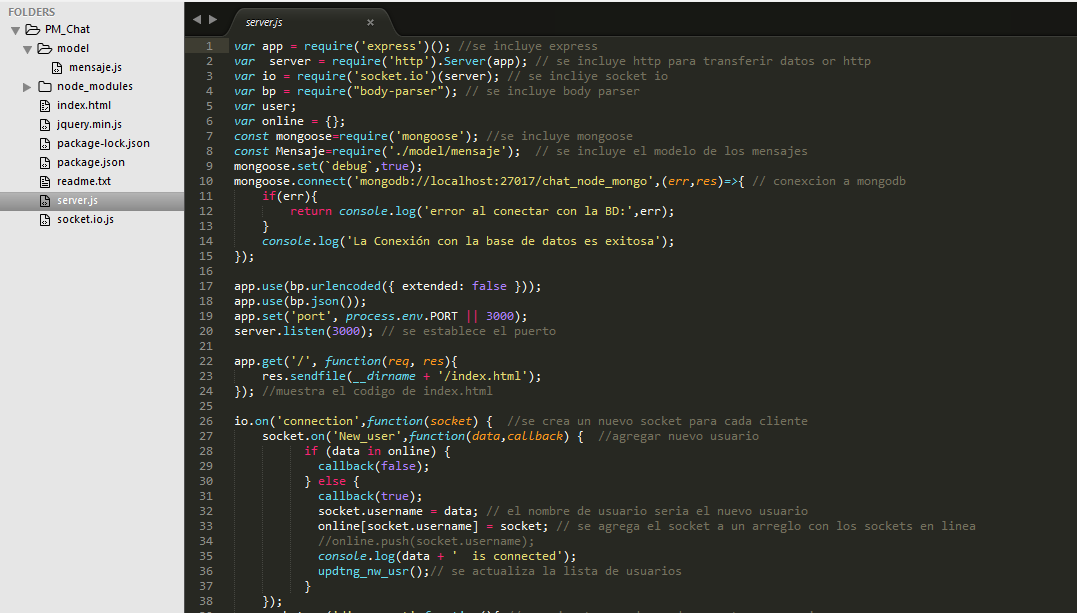


Figura Configuración de la base de datos y puerto.

En la línea 10 a 15 de la figura anterior se establece la conexión con la base de datos, en la línea 20 se establece el puerto por el que está escuchando, y en las líneas a partir de la 26 se crean sockets para cada usuario nuevo que se va conectando y se van actualizando los sockets, los cuales se encuentran dentro de un arreglo.

En la figura 11 se sigue configurando con respecto a los usuarios nuevos, es decir si se desconecta un usuario, se actualiza el socket y muestra solo los usuarios en línea.



Figura Actualización de usuarios.

### Envío de mensajes y guardado en BD.

Ahora para enviar mensajes y recibirlos se usa la función de la figura 12 en el cual se escribe el mensaje y se revisa para quién es (usuario especifico o a todos), si es privado se revisa la lista de usuarios y envía el mensaje al usuario correspondiente, y el mensaje es guardado en la base de datos.



Figura Mandar mensajes privados y guardarlos.

Entonces si el mensaje es para un usuario que esta fuera de línea se recibe “mensaje/notificación” de que el usuario está fuera de línea, por lo cual no se envía el mensaje. Luego está la opción de que el mensaje sea enviado a todos por lo cual solo envía el mensaje (figura 13).



Figura Mensaje para todos y guardado.

Fuera del ciclo anterior se crea otra función para ver si estan escribiendo algún mensaje, y si deja de escribir no muestra mensaje de que estan escribiendo (figura 14).



Figura Muestra quién está escribiendo.

## Primera prueba.

Si con lo que se ha hecho hasta ahora se ejecuta el programa mediante el comando node server.js (porque es el archivo principal donde se encuentra la configuración del lado del servidor) y accediendo a la dirección establecida por el puerto 3000, se puede observar que se muestra la ventana de login (figura 15).



Figura Ventana para loguearse.

En caso de no poner nombre de usuario mandara un error, en este caso es un mensaje solicitando poner un nombre.

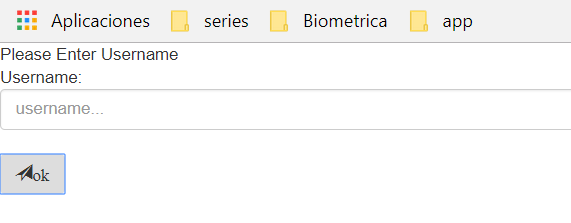


Figura Error al no poner nombre de usuario.

Y en la figura 17 se puede visualizar la pantalla del chat haciendo uso del chat público y privado.

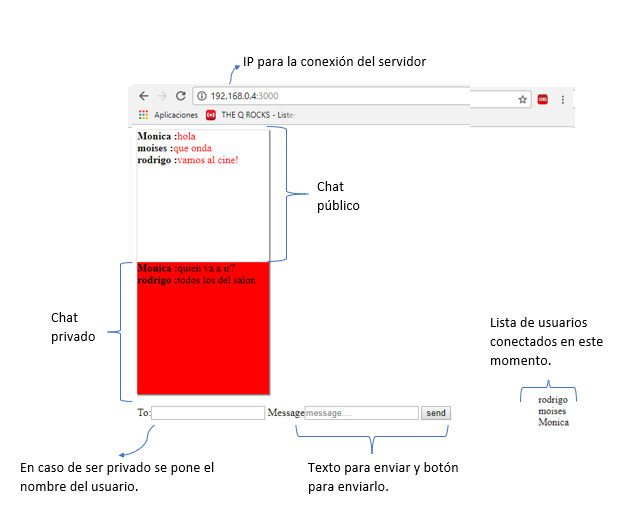


Figura Chats.

Al estar enviando y recibiendo mensajes se debe estar corriendo mongodb para que se vayan guardando los mensajes en la base de datos, en la figura 18 se puede ver cómo se van recibiendo los datos.

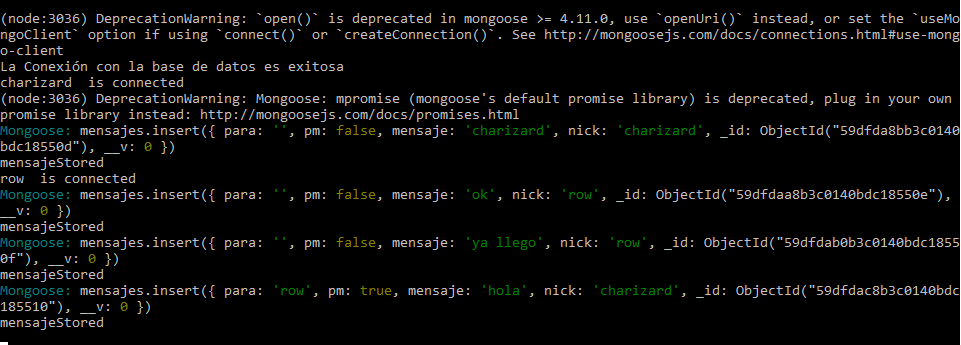


Figura Mongod corriendo en la consola.

Y esos datos que se ven son guardados en la base de datos de mongo en la figura 19 se puede apreciar de manera más clara.

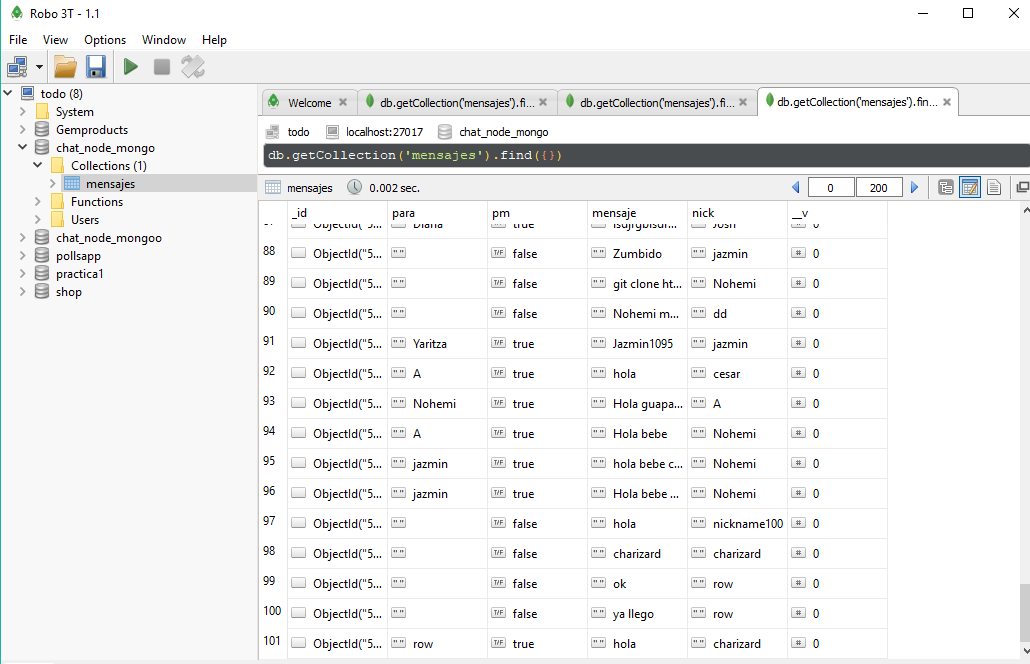


Figura Datos guardados en la base de datos.

Hasta este momento es lo que se tiene, como vemos está funcionando, cumpliendo con su objetivo que es mandar y recibir mensajes, pero la interfaz es muy sencilla, por lo cual más adelante se le modificara y se agregaran otras funcionalidades.

## Funcionalidades extra.

### Zumbido.

Para el envío de zumbidos se agrega un botón dentro del index.html que es del lado del cliente. Lo ponemos dentro del formulario que ya teníamos (figura 20).

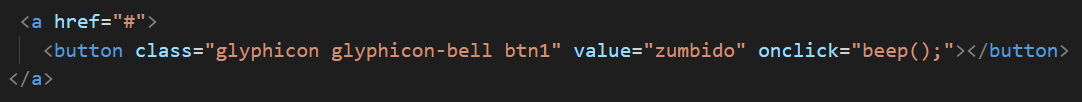


Figura Código para insertar botón.

Dentro de él se encuentra un evento llamado onClick que manda hablar a una función llamada beep, dentro del cliente. Después de ese mismo lado del cliente se agrega la función beep que es la que tiene el audio por así decirlo (figura 21).

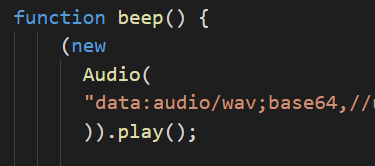


Figura Función beep

Ahora es necesario tener un evento del lado del cliente (figura 22) el cual se envía al servidor.

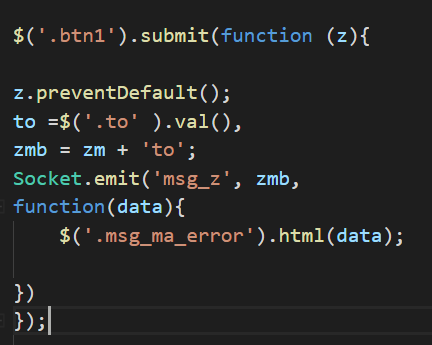


Figura Event lado del cliente,

Y del lado del servidor se tiene que recibir esa instrucción y devolverla (figura 23).



Figura Lado del servidor recibe evento.

Se ejecuta el proyecto para ver los resultados figura 24.

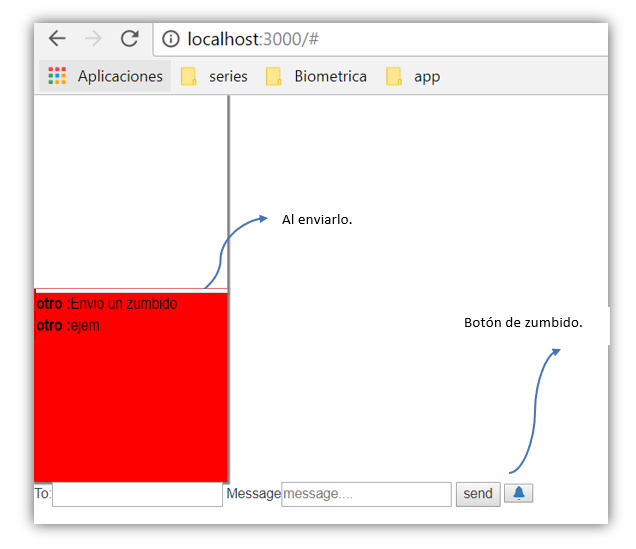


Figura Ejemplo del zumbido

### Estado.

Con esta función se permite ver en el chat público cuando un usuario se conecte a la sala de chat sea comunicado a los demás usuarios ya conectados mediante un mensaje que aparecerá en la caja de chat especificando el usuario y un mensaje diciendo “is Conected”. En el caso contrario cuando un usuario se desconecte, aparecerá en la caja de texto el usuario y el mensaje “is Disconected”, para ello en el lado del servidor dentro del socket new\_user (figura 25) se agregará una línea que emite un mensaje de usuario (nombre) está conectado, esto aparecerá cada que un nuevo usuario entra al chat.



Figura Código de estado conectado.

Al guardar los cambios y ejecutar el programa, cuando un usuario se conecta se muestra como en la figura 26.

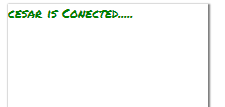


Figura Estado de nuevo usuario.

Ahora para que se muestre el mensaje cuando se desconecte un usuario, se hace dentro del socket de disconnect se agrega la misma línea que en la figura 25 pero ahora el mensaje es “is disconnected” (figura 27).

Figura Código para usuario descoectado

Al guardar los cambios y ejecutar el proyecto se puede ver que cuando un usuario se desconecta, los demás usuarios en el chat reciben el mensaje de que cierto usuario se ha desconectado (figura 28).



Figura Estado cuando usuario deja chat.

### Emojis.

Para los emojis es necesario que dentro de la carpeta content se agreguen los gifs de los que serán los emojis, después agregar un botón del lado del cliente y ahí mismo dentro de index.html (lado del cliente) se generan los eventos (figura 29) para que permitan mostrar los emojis dentro del chat.



Figura Eventos para mostrar los emojis.

Luego anteriormente se mencionó que dentro de la carpeta content se agregaron los ‘emojis’ entonces en la figura 30 se extraen los emojis de la carpeta antes mencionada, lo cual permitirá ver los emojis a mostrar.

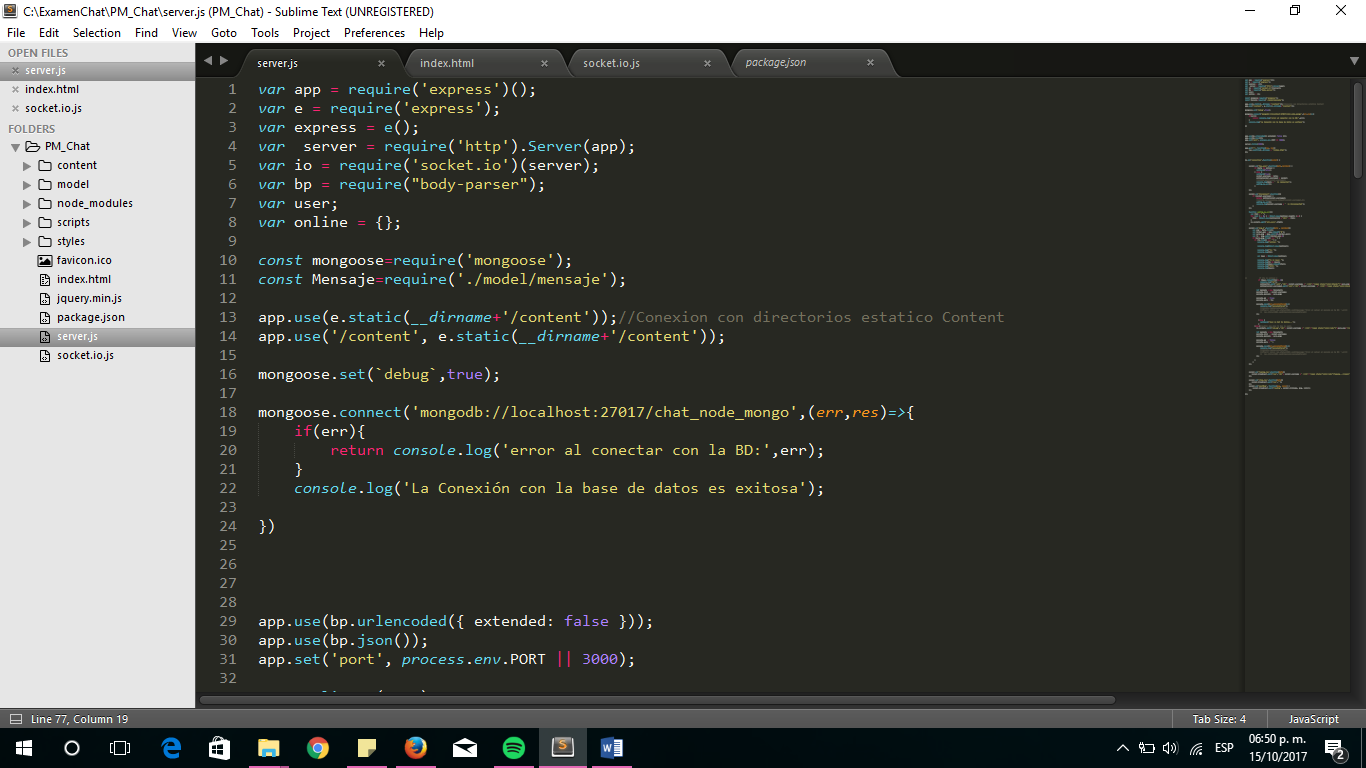


Figura Llamado a los datos de content.

Al ejecutar el proyecto se puede ver que al presionar el botón de emojis se muestran los mencionados (figura 31) y al seleccionarlos se envían.

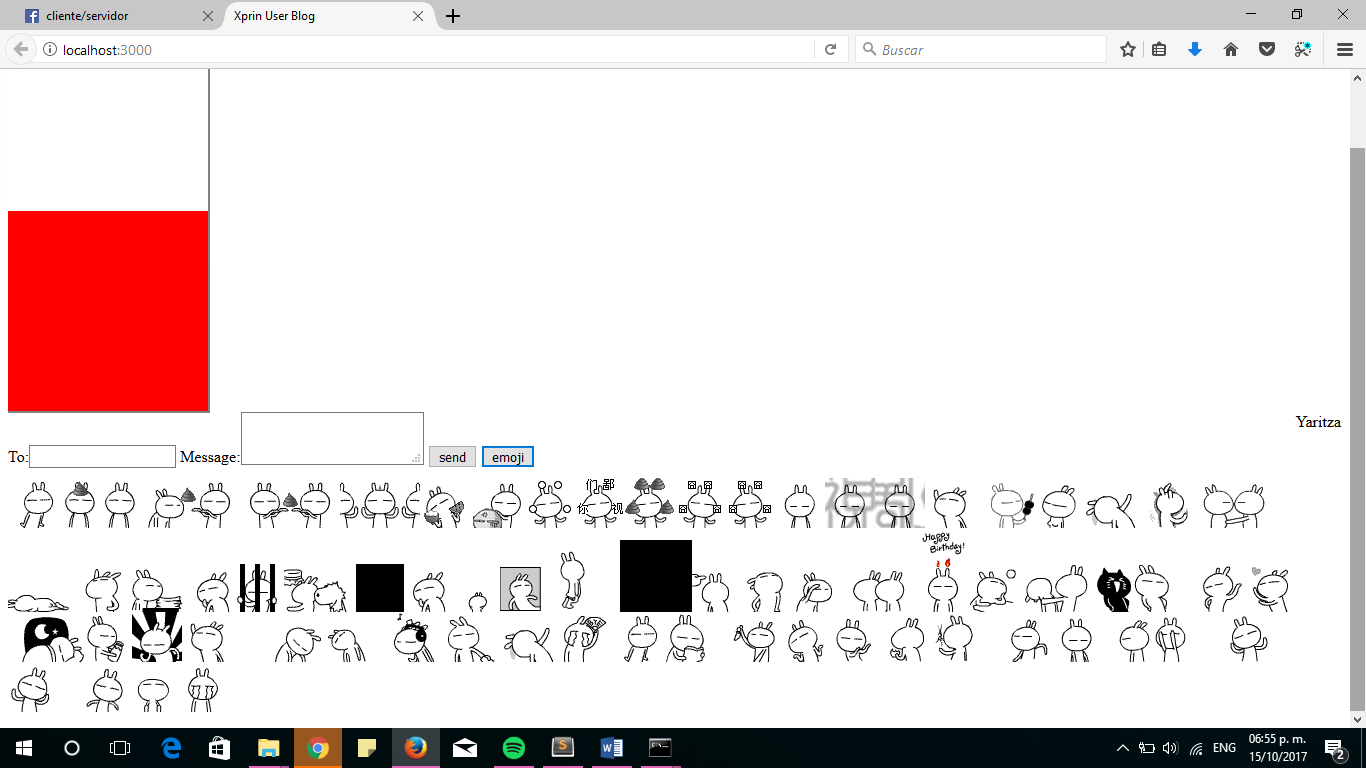


Figura Emojis.

## Vista del chat.

El chat está listo solo falta que la vista sea más agradable para ello se agregaron algunos estilos, los primeros cambios se pueden ver al iniciar el chat en donde tiene que poner un username (figura 32)

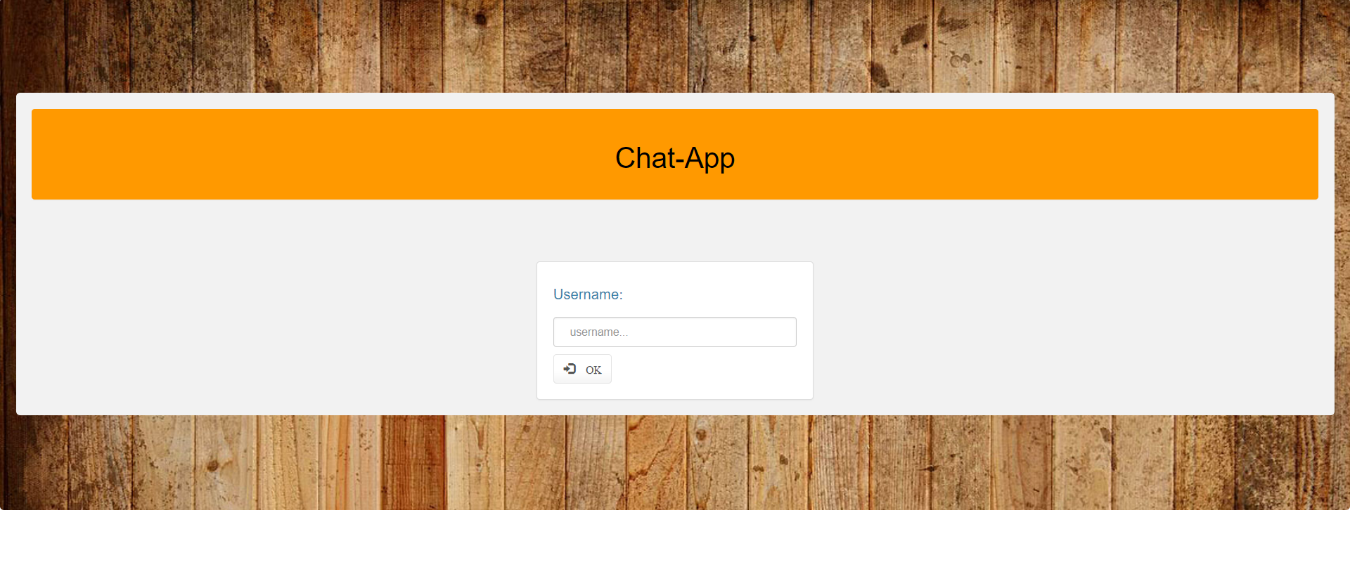
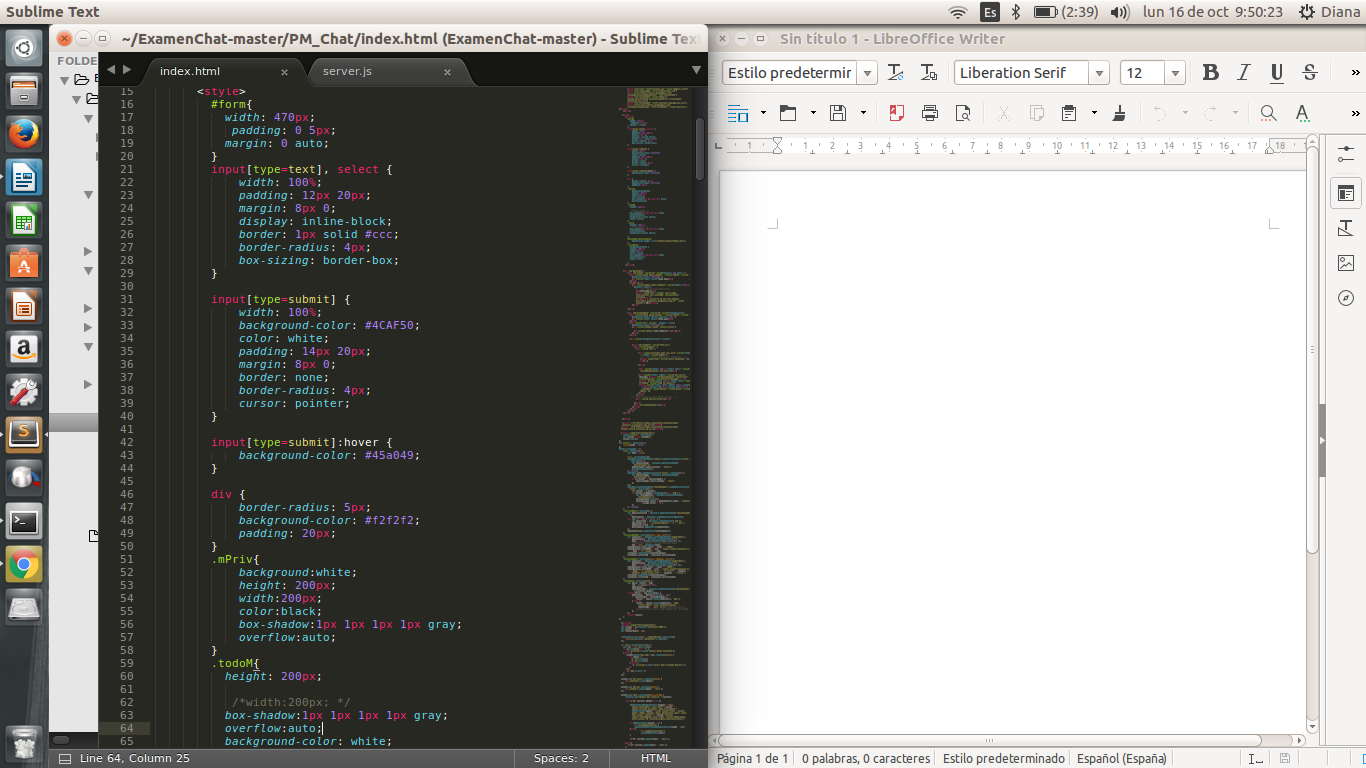


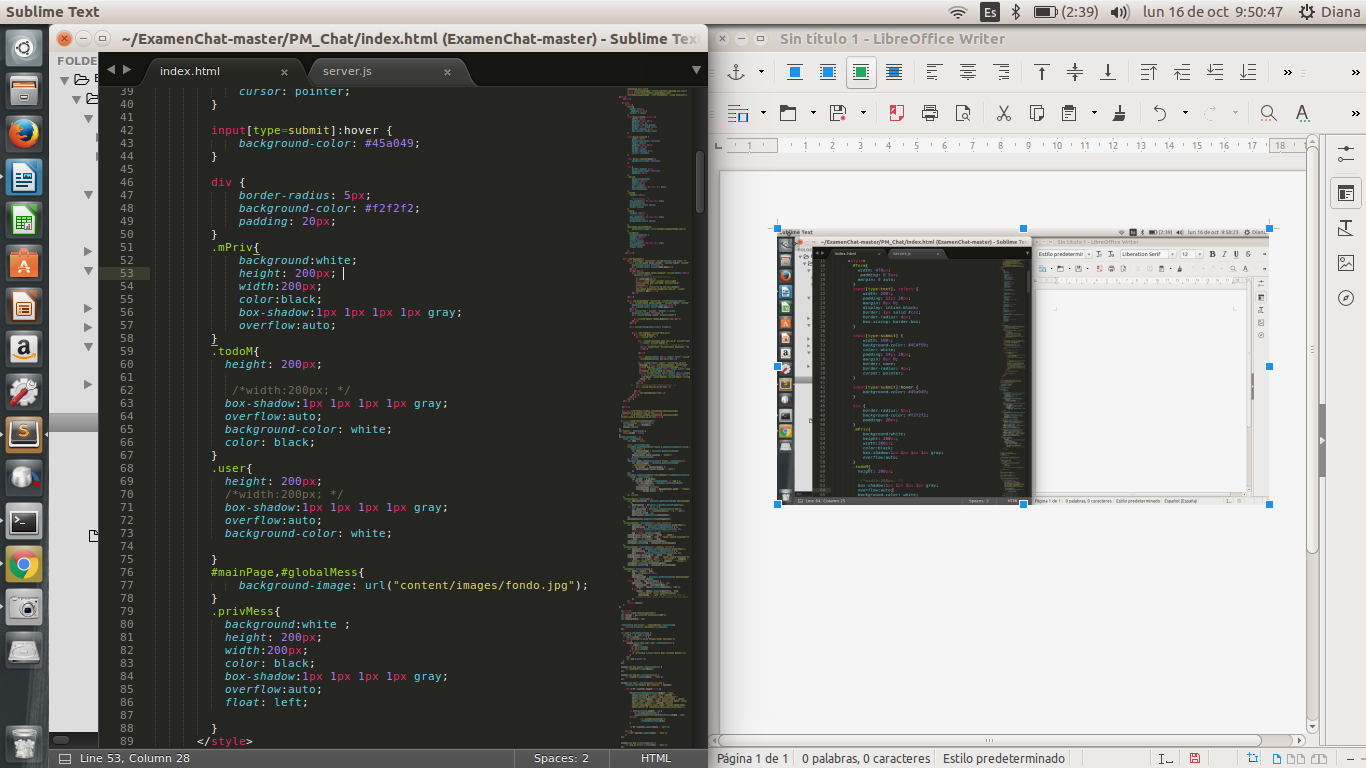
Figura Interfaz de login

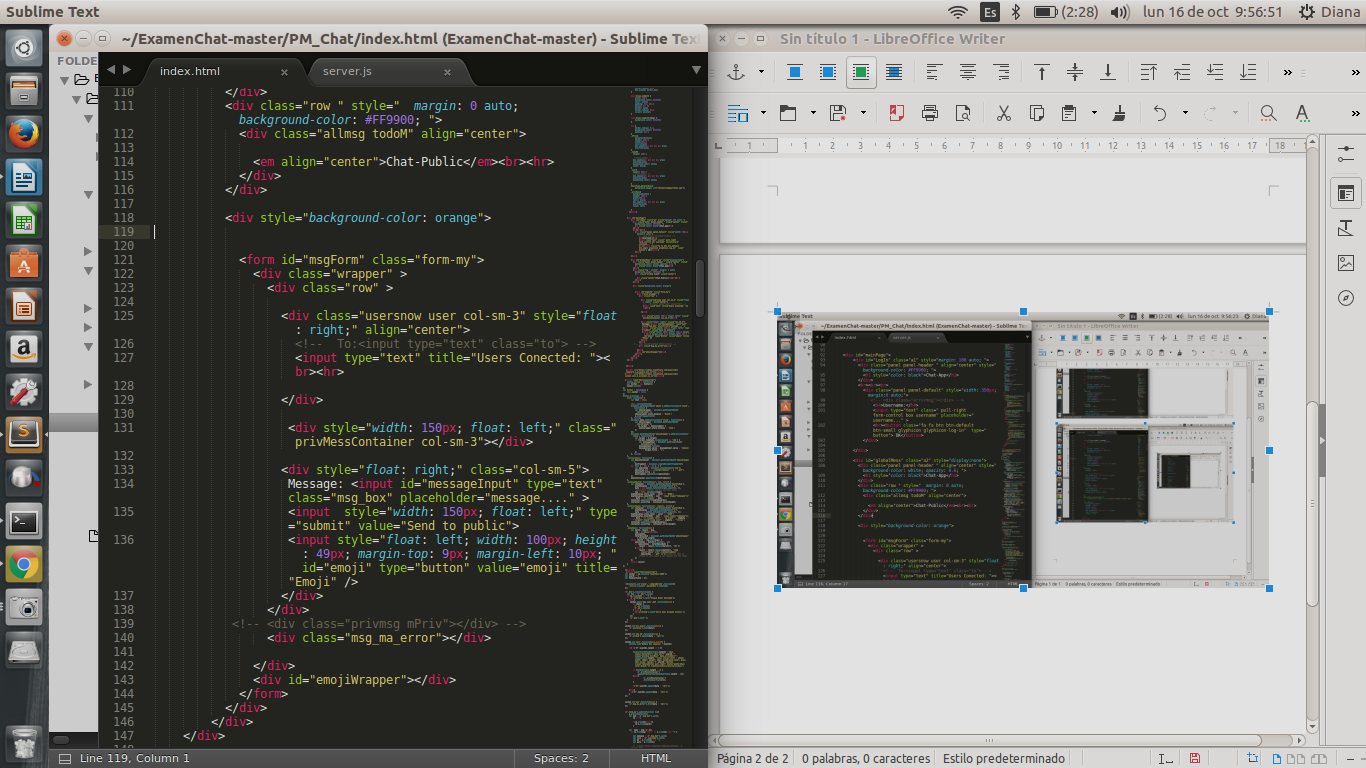
Luego al entrar al chat vemos que la interfaz es más amigable (figura 33) se pueden apreciar de mejor manera lo botones, diferenciar los chats.

Se agregó un estilo al chat en cada una de las partes que complementan la pagina.

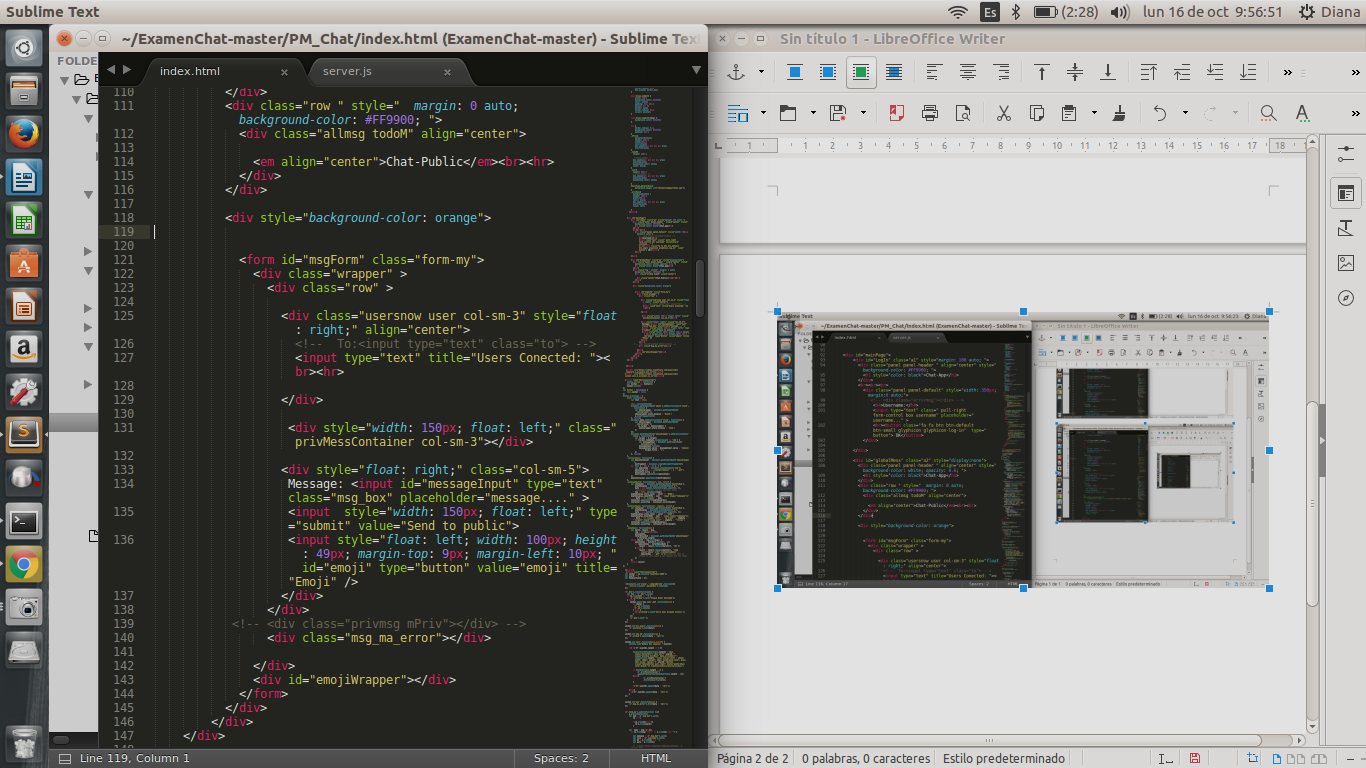
- Los inputs, los div, los botones y todas las etiquetas tienen cada una su estilo,como tambien vienen declarado estilos de etiquetas que tienen su clase o bien id personal.

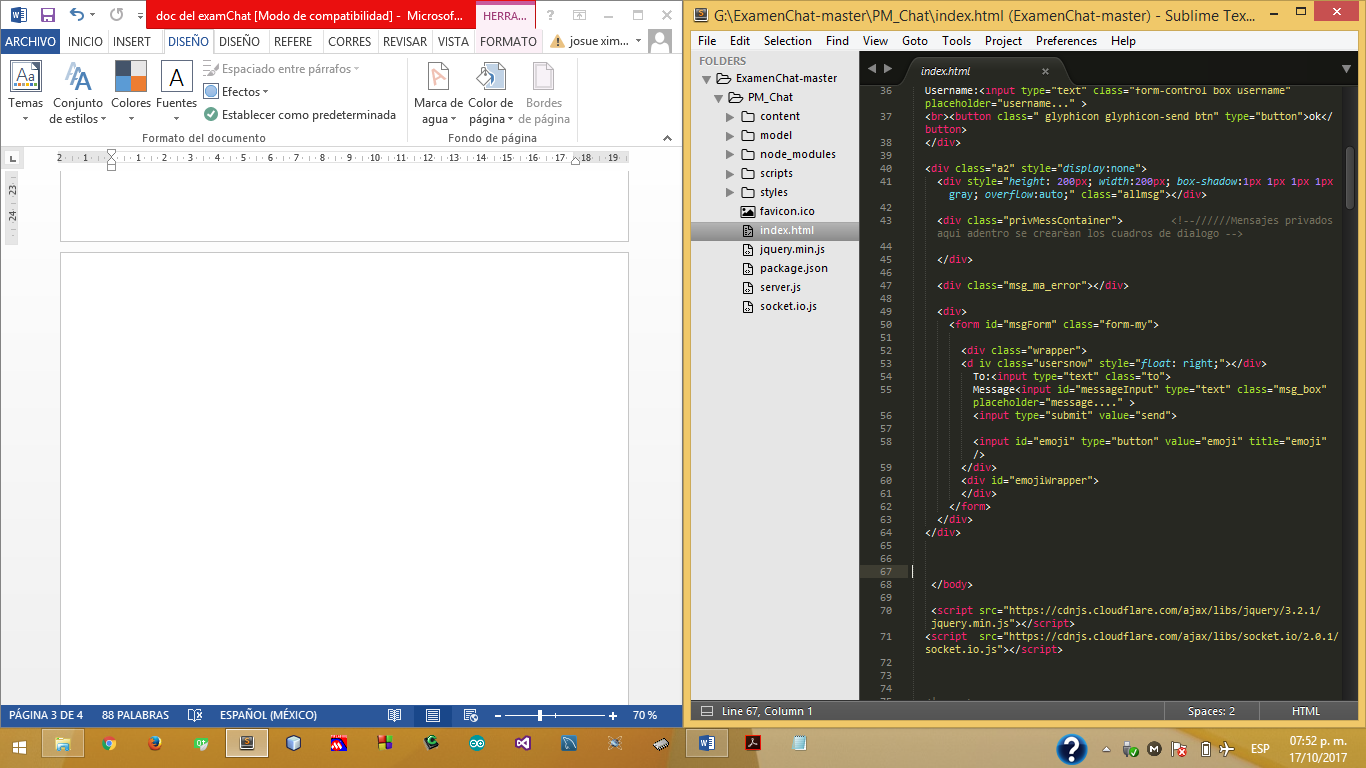






Tambien se agregaron varios div para identificar partes y poder acomodar la primera parte que es el chat publico, segundo es donde se encuentran los chats privados , el envio de texto y los usuarios conectados, tercero, donde marca quien escribe.



1. Para hacer la visualización de usuarios conectados se comenzo modificando con la clase usersnow y creando botones para sumejor visualización eliminando asi la necesidad de tener una etiqueta input para el envio de mensajeria privada ya que se ocacionavan muchos errores al escribir el nombre de los usuarios. Gracias a esto la selección del destio del mensaje es ahora mas facil.
2. En esta función es donde se genera el arreglo de los usuarios conectados y se crean los botones de cada usuario conectado

function updtng\_nw\_usr(){

var html = " ";

for (var i = 0; i < Object.keys(online).length; i++) {

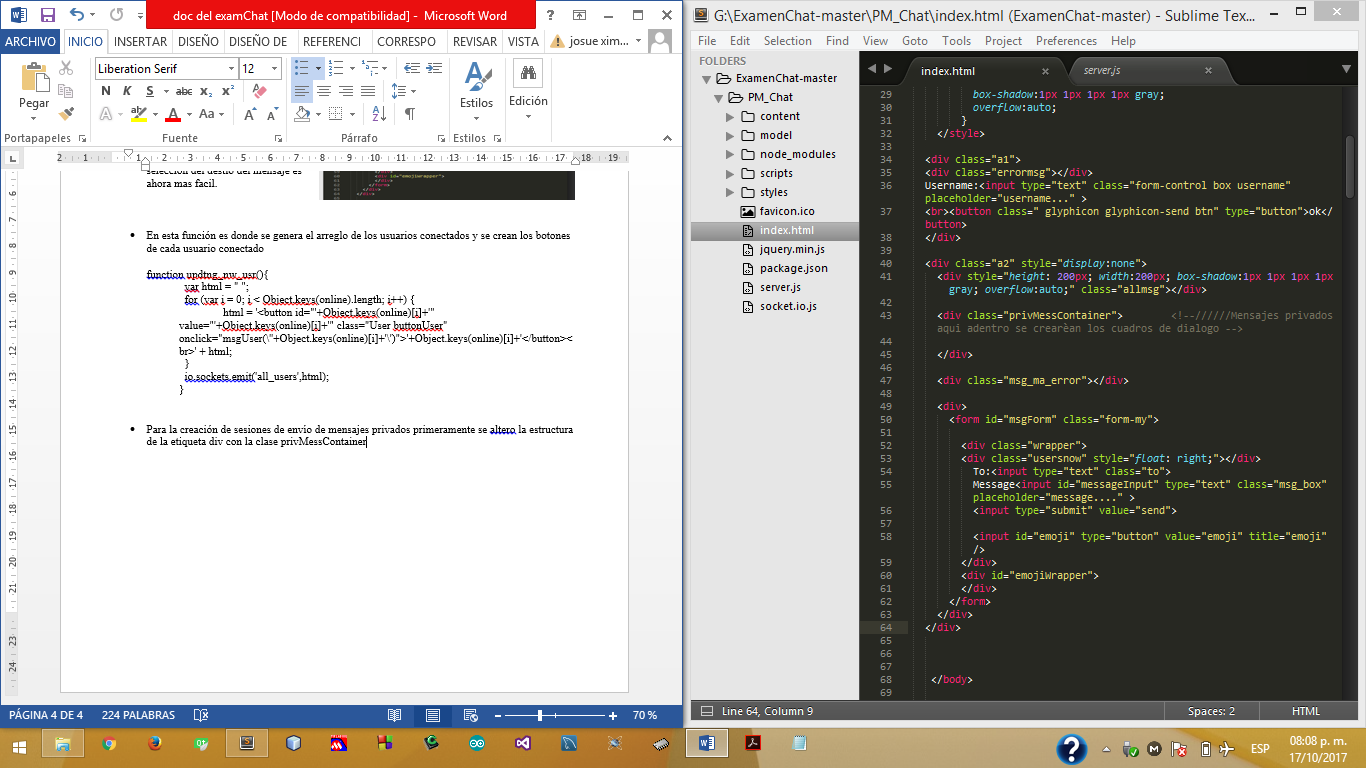
html = '<button id="'+Object.keys(online)[i]+'" value="'+Object.keys(online)[i]+'" class="User buttonUser" onclick="msgUser(\''+Object.keys(online)[i]+'\')">'+Object.keys(online)[i]+'</button><br>' + html;

}

io.sockets.emit('all\_users',html);

}

1. Se creó también la variable: “userM” para el manejo de los usuarios conectados

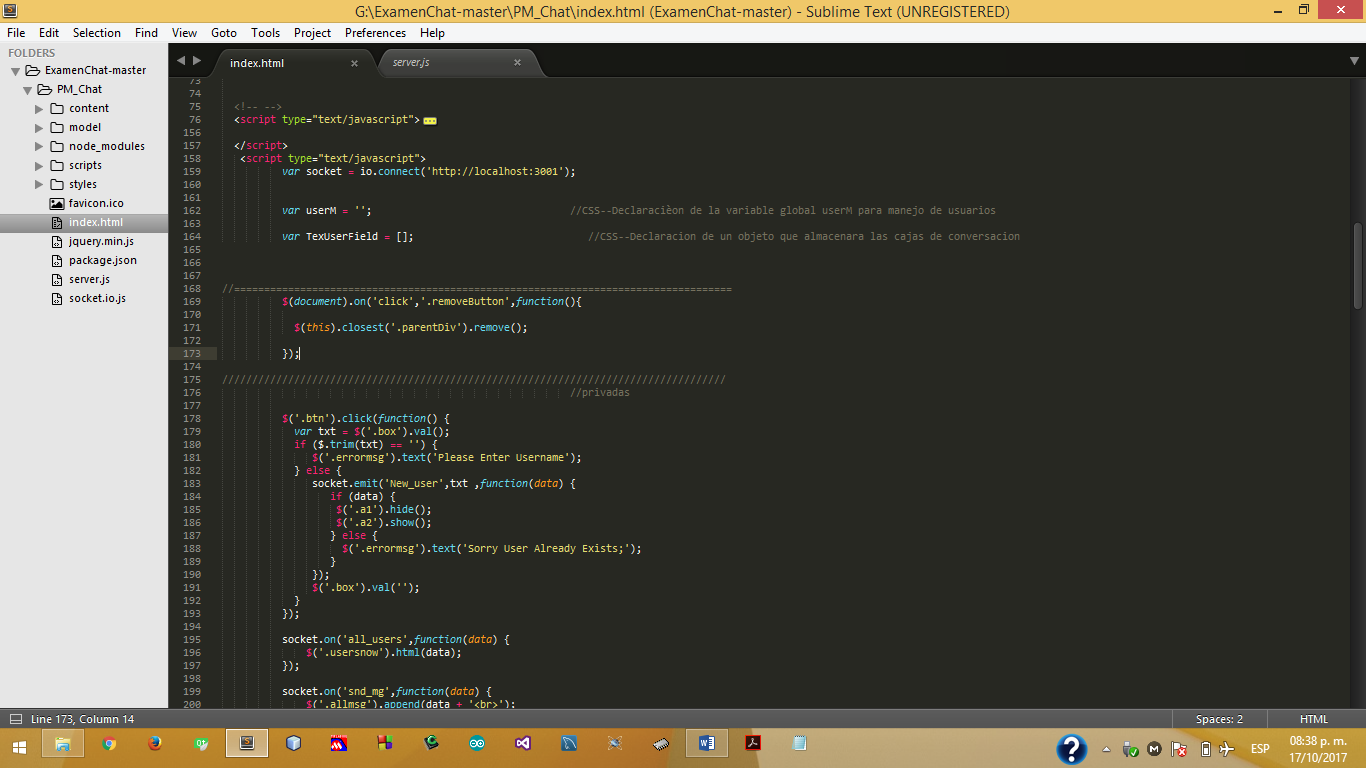
* Para la creación de sesiones de envío de mensajes privados primeramente se altero la estructura de la etiqueta div con la clase privMessContainer
* Seguido se cambió toda la estructura de la función “call” para la creación y envío de los mensajes dependiendo de la sesión creada por los mensajes privados



* Se alteró también la estructura de la función “submit” para detectar si es un mensaje privado o mensaje global. Utilizando la variable userM para saber que usuario envio el mensaje.
* Se creó también una función que crea el submit para el form principal que tiene el ID msgForm
* Se hizo uso de una arreglo para almacenar las seciones creadas por mensajería privad

var TexUserField = []; //CSS--Declaracion de un objeto que almacenara las cajas de conversacion

* Y para remover los contenedores de converacion se genero la siguiente función utilizando los métodos del DOM de HTML



* Por ultimo se modifico la llamada a la función “call” del lado del servidor

