CURSO: CC3S2 – Desarrollo de Software

Lab09 - Programación con NodeJS – y Web Sockets.

Al final entregara un archivo con el nombre **CC3S2-LAB09-<Nombre_apellido>.zip** con los archivos generados por usted.

1. Introducción

Los websockets son una tecnología que permite una comunicación bidireccional entre cliente y servidor sobre un único socket TCP. Con Websockets no tenemos que pedir los datoa para obtenerlos del servidor, ya que el servidor nos los enviará cuando haya nuevos. Uno de los ejemplos más comunes para aprender a utilizar websockets, es desarrollando un chat. Es lo que es lo que veremos en este Lab.

Requisitos: Un servidor de websockets, que construiremos en Node.js y express con la librería Socket.io. Socket.io facilita el desarrollo de aplicaciones utilizando Websockets en el cliente y en el servidor. (Ref: https://socket.io/docs/v3/)

2. Desarrollo.

2.1. Inicializando la Aplicación: Empezamos creando un paquete de Node con *npm init -y* para generar un *package.json* con las opciones por defecto:

```
{
  "name": "lab09-files",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

2.2 Ahora crearemos un archivo server/main.js que será nuestro servidor web. Para ello necesitaremos las librerías express y socket.io que instalamos vía npm :

```
npm install --save express

npm install --save socket.io
```

2.3 En directorio server , en *./server/main.js* creamos una aplicación con express, que pasaremos a un servidor http que se conectará al servidor de websockets que crearemos con socket.io

```
var express = require ('express');
var app = express(); // app es nuestra aplicacion express
var server = require('http').Server(app); // creamos el servidor y le pasamos app
var io = require('socket.io')(server); // al servidor de websockets le pasamos el obj. servidor
```

2.4 Pondremos el servidor http a escuchar en localhost con el puerto 8080 :

```
server.listen(8080, function() {
console.log('Servidor corriendo en http://localhost:8080');
});
```

Probamos que el sevidor esté funcionando con
`node ./server/main.js.`

2.5 Para agilizar el proceso de desarrollo instalamos nodemon que se encarga de re-inicializar el servidor NodeJS cada vez que hagamos un cambio en nuestra aplicación.
`npm install nodemon --save`
y luego editamos el archivo package.json en la seccion scripts:

```
"scripts": {
"start": "nodemon ./server/main.js"
},
```

2.6 En `main.js` Agregamos una ruta a nuestro servidor web para que nos muestre la pagina por defecto:
antes de server.listen…

```
app.get('/', function(req,res){
res.status(200).send("Estamos respondiendo!");
});
```

Esta vez probamos que el sevidor esté funcionando con *npm start*

2.8 Configuramos el servidor de webSockets para que responda a eventos de conexión: con una función que recibe como parámetro el socket que se esta utilizando para la conexión

```
io.on('connection', function(socket){
        console.log('Alguien Se ha conectado con sockets');
});
```

2.9 Para probar la conexión a nuestro servidor de websockets necesitamos un cliente web (Javascript) que envie los mensajes y se conecte. Para ello creamos la parte publica (estatica) de la aplicación.

Creamos una carpeta en la raiz *public,* un archivo *index.html* y un archivo *main.js* dentro de ella.

2.10 para poder usar la parte estática de nuestra aplicación utilizamos el metodo static del framework express, indicandole el directorio que será publico. Para ello en el archivo main.js de la parte servidor agregamos la siguiente linea:

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

2.11 Luego, implementamos *index.html de la parte publica* de la siguiente manera: (socket.io es una librería que se utilizará tanto en el navegador como en el servidor de websockets.)

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<title>Mi aplicacion con sockets</title>
	<script src="/socket.io/socket.io.js"></script>
	<script src="main.js"></script>
</head>
<body>
	<h1>Mi Aplicacion con sockets</h1>

</body>
</html>
```

I
2.12 implementamos el archivo , main.js de la parte cliente o publica:
creando un variable socket que utiliza io (que es de la librería socket.io del lado cliente) para que se conecte al servidor websockets que esta corriendo en http://localhost:8080

```
var socket = io.connect('http://localhost:8080',{'forceNew': true});
```

Al verificar la pagina http://localhost:8080/index.html
veremos en la consola lo siguiente:
Servidor corriendo en http://localhost:8080
Alguien Se ha conectado con sockets

2.13 Agregamos el código para que podamos recibir mensajes:
Para recibir mensajes utilizamos el metodo *socket.on()* para escuchar el evento *mensajes*, de la siguiente manera: en main.js de la parte cliente:

```
socket.on('mensajes',function(data){
	console.log('Se ha recibido: ', data );
});
```

2.14 Para probar que en el cliente se puede recibir mensajes los enviaremos desde el servidor. Para ello, en el archivo main.js del lado servidor: modificamos las siguientes lineas en la función callback que se utiliza para las conexiones para emitir eventos de tipo *mensajes* :

```
io.on('connection', function(socket){
        console.log('Alguien Se ha conectado con sockets');
        socket.emit('mensajes',
        {
                id: 1,
                author: "Pedro",
                text: "Hola Sr. Rajuela!"
        });
});
```

y verificamos en la consola del navegador si se ha recibido la información.

2.15 Para mostrar los mensajes que llegan del servidor en la página web en vez de en la consola del browser, hacemos las siguientes modificaciones:

En el documento index.html agregamos un para escribir los mensajes.

```
<body>
        <h1>Mi Aplicacion con sockets</h1>
        <div id="mensajes"></div>
</body>
```

y en el archivo main.js de la parte cliente, agregamos una función *render()* para que muestre el mensaje

```
        var socket = io.connect('http://localhost:8080',{'forceNew': true});
        socket.on('mensajes',function(data){
                console.log('Se ha recibido: ',data );
                render(data);
        });

function render(data){
var html=`
        <div>
                <strong>${data.author} </strong>
                <em>${data.text}</em>
        </div>`;
document.getElementById('mensajes').innerHTML=html;
}
```

Verificamos en el browser que se muestre el mensaje enviado desde el servidor.

2.16 Ahora enviaremos un mensaje desde el cliente. Para ello modificamos el documento index.html agregando un formulario para enviar el mensaje, que llame a una función addMessage() en el evento onSubmit del formulario. AddMessage() seá definida en el Javascript del lado cliente, para conectarse al servidor webSockets.

```
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Mi aplicacion con sockets</title>
        <script src="/socket.io/socket.io.js"></script>
        <script src="main.js"></script>
</head>
<body>
        <h1>Mi Aplicacion con sockets</h1>
        <div id="mensajes"></div>
        <form onsubmit="return addMessage(this)">
                <input type="text" id="username" placeholder="Tu Nombre">
```

```
            <input type="text" id="texto" placeholder="En que estas pensando?...">
            <input type="submit" value="Enviar!">
        </form>
</body>
```

En el archivo *main.js de la parte cliente* creamos la función addMessage(): que generará un evento
'new-message', enviando la información del formulario.

```
function addMessage(e){
        var payload={
            author: document.getElementById("username" ).value,
            text: document.getElementById("texto" ).value
        }
        socket.emit('new-message',payload);
        return false;
}
```

2.17 En el lado servidor en el archivo *main.js* empezaremos a escuchar eventos del tipo 'new-message' para ello modificamos la funcion *io.on()…* para que envie un arreglo de mensajes 'messages' y pueda guardar en ese arreglo los mensajes que puedan llegar al servidor. Creamos el arreglo messages para las pruebas.

```
var messages=[
        {
            author: "Pedro",
            text: "Hola Sr. Rajuela!"
        },
        {
            author: "Sr. Rajuela",
            text: "Hola Pedro! que haces por aquí?"
        },
        {
            author: "Pablo",
            text: "Me parece que iré por unas Bronto!"
        },
        {
            author: "Pedro",
            text: "Genial!"
        }
]
io.on('connection', function(socket){
        console.log('Alguien Se ha conectado con sockets');
        socket.emit('mensajes',messages);
});
```

2.18 Ahora en el lado cliente en main.js, modificamos la función render(data) para que pueda recibir un arreglo de mensajes

```
function render(data){
        var html= data.map(function(elem,index){
            return(`<div>
                    <strong>${elem.author} </strong>
                    <em>${elem.text}</em>
                    </div>`)
```

```
                }).join(" ");
        document.getElementById('mensajes').innerHTML=html;
}
```

Podemos verificar que el arreglo esta llegando al cliente en el browser.

2.19 En el lado servidor en main.js terminamos de modificar *io.on ()* para que pueda recibir los mensajes en el arreglo *messages* del cliente escuchando eventos 'new-message' que provienen del cliente. Ademas de guardar los mensajes entrantes en el arreglo, el servidor *io.sockets.emit('mensajes', messages)* enviará los mensajes a todos los clientes conectados.

```
io.on('connection', function(socket){
        console.log('Alguien Se ha conectado con sockets');
        socket.emit('mensajes',messages);

        socket.on('new-message',function(data){
        messages.push(data);
        socket.emit('mensajes',messages);
        })
});
```

2.20 Puede probar conectandose con 2 clientes a la vez y observar que se actualizan los mensajes en todos los clientes.

3. Tarea:

Implementar un cliente de chat para su servidor con ReactJS y Material-UI .
Presentar un reporte **CC3S2-LAB09-<Nombre_apellido>.zip** con los screenshots y los archivos generados del su aplicación. Excluya el directorio node_modules.