Universidad Autónoma Juan Misael Saracho
Facultad de Recursos Naturales y Tecnología
Carrera de Ingeniería Informatica

INFORME "CHAT CON NOTIFICACIONES" Y "JUEGO PIEDRA, PAPEL Y TIJERA" USANDO SOCKETS

MATERIA : Tecnologia de Programación en Red
SIGLA : IEF423
DOCENTE : Ing Elias Cassal Baldiviezo
ESTUDINATE : José Eduardo Reyes Flores
Rey Jesús Zeballos Lopez

22/10/2024

YACUIBA-TARIJA

Informe: Chat con notificaciones usando WebSockets y Socket.IO

Introducción

Este proyecto implementa una aplicación de chat en tiempo real utilizando Node.js y Socket.IO. La aplicación permite a los usuarios conectarse al servidor, enviar y recibir mensajes de chat, notificar cuando un usuario está escribiendo, y gestionar las conexiones y desconexiones. Los mensajes y las interacciones entre los usuarios se transmiten en tiempo real utilizando WebSockets.

Estructura del Proyecto

La estructura del proyecto incluye los siguientes archivos:

- **package.json**: Define las dependencias del proyecto y los scripts para ejecutar el servidor.
- **Package-lock.json**: Actúa como un registro detallado de las dependencias instaladas, incluyendo las versiones exactas y las subdependencias.
- **index.js**: Es el archivo principal del servidor que configura la aplicación de Express y Socket.IO.
- **public/index.html**: La interfaz de usuario donde los usuarios pueden enviar mensajes y ver las interacciones en el chat.
- **public/chat.js**: Archivo JavaScript que gestiona la interacción en el frontend con el servidor y maneja las interacciones del usuario.
- **public/main.css**: Archivo para los estilos.
- **public/sonidos**: Sonidos para notificaciones de mensajes, desconexión y cuando un usuario está escribiendo.

Paso 1: Instalación de Dependencias

Para que el proyecto funcione, necesitas instalar las siguientes dependencias:

1. express: Framework de servidor para Node.js.
2. socket.io: Librería para la comunicación en tiempo real a través de WebSockets.
3. nodemon: Herramienta para reiniciar automáticamente el servidor cuando se detecten cambios en el código (opcional, pero útil para desarrollo).

Para instalar estas dependencias, sigue estos pasos:

```
npm init -y # Inicializa un proyecto de Node.js
npm install express # Instala express
npm install socket.io # Instala socket.io
npm install nodemon --save-dev # Instala nodemon para desarrollo
```

Después de instalar las dependencias, se pueden usar los siguientes scripts en el package.json para iniciar el servidor:

- **npm start**: Para ejecutar el servidor sin recarga automática.
- **npm run dev**: Para ejecutar el servidor con recarga automática usando **nodemon**. Luego en algún navegador ingresar la ruta “localhost:3000”

Paso 2: Configuración del Servidor en index.js

En el archivo index.js, el servidor se configura con Express para gestionar las rutas estáticas y Socket.IO para manejar las conexiones WebSocket de los clientes.

Código de index.js

```
const path = require('path');
const express = require('express');
const app = express();
//settings
app.set('port', process.env.PORT || 3000);
//static files
app.use(express.static(path.join(__dirname, 'public')));
//start the server
const server = app.listen(app.get('port'), () => {
  console.log('server on port', app.get('port'));
});
//websockets
const SocketIO = require('socket.io');
const io = SocketIO(server);
// almacenar los nombres de usuarios
const users = {};
io.on('connection', (socket) => {
  console.log('new connection', socket.id);
// Escuchar cuando un usuario envía su nombre
  socket.on('chat:register', (username) => {
    users[socket.id] = username; // Asociar el username al socket.id
    console.log(` ${username} se ha conectado `);
  });
```

```
  // Emitir mensaje a todos los usuarios, incluyendo al emisor
  socket.on('chat:message', (data) => {
    io.sockets.emit('chat:message', data);
    socket.broadcast.emit('chat:notification', {
      message: `${data.username} envió un mensaje`
    });
  });
  // Cuando alguien está escribiendo, emitir a los demás
  socket.on('chat:typing', (data) => {
    socket.broadcast.emit('chat:typing', data);
    socket.broadcast.emit('chat:notification', {
      message: `${data} está escribiendo...`
    });
  });
  // Detectar cuando un usuario se desconecta
  socket.on('disconnect', () => {
    const username = users[socket.id] || 'Alguien'; // Recuperar el nombre del usuario
    delete users[socket.id]; // Eliminar del registro
    console.log(`${username} se ha desconectado`);
    io.sockets.emit('chat:notification', {
      message: `${username} se ha desconectado`,
      type: 'disconnect'
    });
  });
});
```

Explicación del código:

1. **Configuración del servidor Express**:
    - El servidor se configura para escuchar en el puerto 3000 (o el puerto especificado en el entorno).
    - La carpeta public se usa para servir archivos estáticos, como index.html y chat.js.

2. **Configuración de Socket.IO**:
    - Se inicializa Socket.IO sobre el servidor Express.
    - Se escucha el evento connection cuando un cliente se conecta al servidor.
3. **Manejo de Eventos**:
    - **chat:register**: Cuando un usuario se registra, se guarda su nombre de usuario asociado a su socket.id en un objeto users.
    - **chat:message**: Se emite el mensaje a todos los clientes conectados, incluyendo al emisor.
    - **chat:typing**: Notifica a los demás usuarios cuando alguien está escribiendo.
    - **chat:notification**: Se emiten notificaciones, por ejemplo, cuando alguien se desconecta.
    - **disconnect**: Se maneja la desconexión de un usuario, eliminando su nombre del objeto users y notificando a los demás usuarios.

Paso 3: Configuración del Frontend (HTML y JavaScript)

El archivo index.html contiene la interfaz del usuario, donde los usuarios pueden ingresar su nombre y mensaje para el chat. Además, se incluye el archivo chat.js, que maneja la interacción con el servidor.

Código de index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://fonts.googleapis.com/css2?family=Nunito&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="main.css">
  <title>Chat Socket.io</title>
</head>
<body>
  <div id="chat-container">
    <div id="chat-window">
      <div id="output"></div>
      <div id="actions"></div>
    </div>
```

```
    <div id="notification"></div>
    <script src="/socket.io/socket.io.js" charset="UTF-8"></script>
    <script src="chat.js" charset="UTF-8"></script>
  </body>
  </html>
```

Código de chat.js

```
const socket = io();
let message = document.getElementById('message');
let username = document.getElementById('username');
let btn = document.getElementById('send');
let output = document.getElementById('output');
let actions = document.getElementById('actions');
let notification = document.getElementById('notification');
// instancias de Audio
const sonidoMensaje = new Audio('/sonidos/mensaje.mp3');
const sonidoDesconexion = new Audio('/sonidos/desconexion.mp3');
const sonidoEscribir = new Audio('/sonidos/escribir.mp3');

let escribiendoActivo = false;
username.addEventListener('change', function () {
    socket.emit('chat:register', getUsername());
});
btn.addEventListener('click', function () {
    const user = getUsername();
    socket.emit('chat:message', {
        message: message.value,
        username: user,
        socketId: socket.id
    });
```

```
    message.value = '';
});
message.addEventListener('keypress', function () {
    const user = getUsername();
    socket.emit('chat:typing', user);
    if (!escribiendoActivo) {
        sonidoEscribir.currentTime = 0;
        sonidoEscribir.play();
        escribiendoActivo = true;
    }
});
message.addEventListener('keyup', function () {
    if (message.value === '') {
        socket.emit('chat:stopTyping', getUsername());
    }
});
socket.on('chat:message', function (data) {
    actions.innerHTML = '';
    output.innerHTML += `<p><strong>${data.username}</strong>: ${data.message}</p>`;
    if (data.socketId !== socket.id) {
        showNotification(`${data.username} envió un mensaje`, sonidoMensaje, 'green');
    }
});
socket.on('chat:typing', function (data) {
    actions.innerHTML = `<p><em>${data} está escribiendo...</em></p>`;
    showNotification(`${data} está escribiendo...`, sonidoEscribir, 'yellow');
});
socket.on('chat:stopTyping', function () {
    actions.innerHTML = '';
});
```

```
socket.on('chat:notification', function (data) {
    if (data.type === 'disconnect') {
        showNotification(data.message, sonidoDesconexion, 'red');
    }
});
function getUsername() {
    return username.value || 'Usuario Desconocido';
}
function showNotification(message, sonido, color) {
    notification.innerHTML = message;
    notification.style.color = color;
    sonido.currentTime = 0;
    sonido.play();
}
```

Explicación del código:

- chat.js gestiona la lógica para interactuar con el servidor mediante eventos Socket.IO.
- **chat:register**: El nombre de usuario se emite al servidor cuando el campo de texto cambia.
- **chat:message:** Cuando se envía un mensaje, se emite al servidor con el contenido y el nombre del usuario.
- **chat:typing y chat:stopTyping:** Se emite un evento cuando un usuario comienza a escribir y cuando deja de escribir.
- **chat:notification**: Este evento maneja notificaciones generales como desconexiones.
- **showNotification:** Muestra notificaciones en pantalla y reproduce un sonido asociado. Cambia el color del texto de la notificación según el tipo de mensaje (verde para mensajes, amarillo para escritura y rojo para desconexiones).
- **Sonidos de notificación**: Se gestionan tres sonidos: para cuando llega un mensaje, cuando alguien se desconecta y cuando alguien está escribiendo.

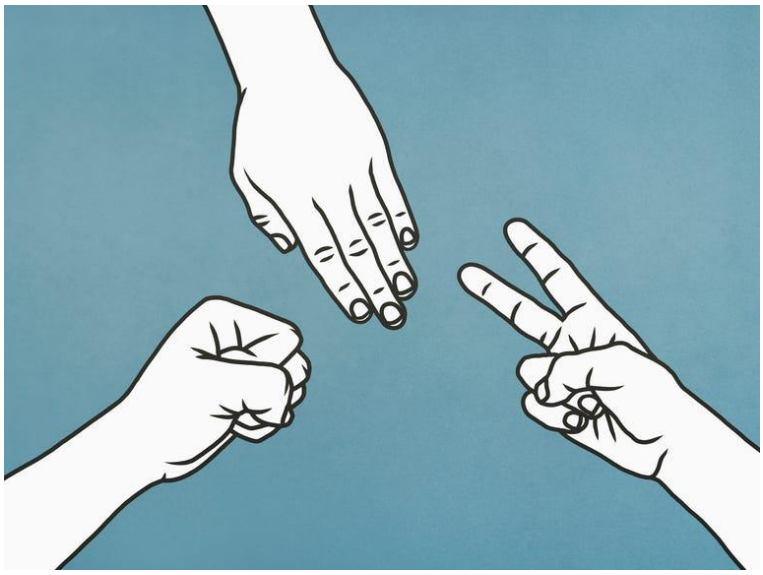
Conclusión

La aplicación de chat desarrollada permite una comunicación en tiempo real entre usuarios utilizando WebSockets y **Socket.IO**. Este sistema se beneficia de la capacidad de **WebSockets** para enviar mensajes de forma bidireccional y sin necesidad de recargar la página, lo que facilita interacciones fluidas y rápidas. Además, la implementación incluye características de notificación y de "escribiendo..." que mejoran la experiencia del usuario en chats en vivo.

Informe: Juego "Piedra, Papel o Tijera" con Sockets

Introducción

El proyecto "Piedra, Papel o Tijera" es un sistema interactivo basado en web que permite a dos jugadores competir en tiempo real. Utiliza tecnologías modernas como Node.js, Express, y Socket.IO, junto con un diseño visual atractivo mediante HTML, CSS y JavaScript. Este informe describe en detalle su funcionalidad, arquitectura, dependencias, y proceso de desarrollo, proporcionando una visión integral del proyecto.



Estructura del Proyecto

El proyecto se organiza en las siguientes carpetas y archivos clave:

- **public/**: Contiene los archivos estáticos accesibles desde el navegador.
    - **index.html:** La interfaz principal del juego.
    - **styles.css:** Define los estilos visuales.
    - **script.js:** Gestiona los eventos en el cliente.
    - **images/**: Carpeta con la imagen de fondo (fondo.png).
- **Archivo del servidor:**
    - **server.js:** Código del backend que maneja la lógica del juego y la comunicación entre jugadores.
- **Archivos de configuración:**
    - **package.json:** Especifica las dependencias del proyecto y comandos de inicio.
    - **package-lock.json:** Contiene las versiones específicas de las dependencias para asegurar compatibilidad.

Configuración del Proyecto

Archivo package.json

Este archivo configura el entorno del proyecto y define los paquetes utilizados, junto con los scripts para facilitar su ejecución.

```
{
  "name": "nuevo",
  "version": "1.0.0",
  "main": "script.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.21.1",
    "socket.io": "^4.8.1"
  }
}
```

Dependencias Clave:

1. **Express**: Maneja la creación del servidor y el manejo de archivos estáticos.
2. **Socket.IO**: Permite la comunicación en tiempo real entre el cliente y el servidor.

Archivo package-lock.json

- Este archivo se genera automáticamente al instalar dependencias y asegura que las versiones exactas utilizadas sean compatibles entre entornos. Contiene metadatos detallados sobre las dependencias.

Funcionalidad del Juego

Flujo del Juego:

1. Los jugadores acceden al sistema desde el navegador.
2. Un jugador realiza su jugada seleccionando Piedra, Papel, o Tijera.
3. El servidor espera la jugada del segundo jugador.
4. Al recibir ambas jugadas, el servidor calcula el resultado y actualiza el historial.
5. Ambos jugadores reciben el resultado y el historial actualizado.

Backend: Servidor (server.js)

El archivo server.js contiene toda la lógica de backend, incluidas las siguientes funcionalidades:

- Manejo de conexiones con **Socket.IO**.
- Almacenamiento temporal de jugadas de los jugadores.
- Resolución de partidas según las reglas de **Piedra, Papel o Tijera**.
- Envío de resultados e historial a los jugadores.

Código de server.js

```
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');
const app = express();
const server = http.createServer(app);
const io = new Server(server);
let players = {};
let history = [];
app.use(express.static('public'));
io.on('connection', (socket) => {
  console.log(` Jugador conectado: ${socket.id}`);
  socket.on('play', (move) => {
    players[socket.id] = move;
    if (Object.keys(players).length === 2) {
      const [player1, player2] = Object.keys(players);
      const result = resolveGame(players[player1], players[player2]);
```

```
history.push({
            player1: { id: player1, move: players[player1], result: result.player1 },
            player2: { id: player2, move: players[player2], result: result.player2 }
          });
          io.to(player1).emit('result', { result: result.player1, history });
          io.to(player2).emit('result', { result: result.player2, history });
          players = {};
        }
    });
    socket.on('disconnect', () => {
        console.log(`Jugador desconectado: ${socket.id}`);
        delete players[socket.id];
    });
});
function resolveGame(move1, move2) {
    if (move1 === move2) return { player1: 'Empate!', player2: 'Empate!' };
    if (
        (move1 === 'piedra' && move2 === 'tijera') ||
        (move1 === 'papel' && move2 === 'piedra') ||
        (move1 === 'tijera' && move2 === 'papel')
    ) {
        return { player1: 'Ganaste!', player2: 'Perdiste!' };
    } else {
        return { player1: 'Perdiste!', player2: 'Ganaste!' };
    }
}
server.listen(3000, () => {
    console.log('Servidor corriendo en http://localhost:3000');
});
```

Frontend

Archivo index.html

Define la estructura del juego, incluyendo:

- Una ventana modal para las partidas.
- Botones interactivos para elegir la jugada.
- Un área dinámica para mostrar resultados e historial.

Archivo script.js

Implementa la lógica del cliente:

- Conexión al servidor con **Socket.IO**.
- Envío de la jugada del jugador.
- Actualización del historial y resultados.

Archivo styles.css

Aplica un diseño atractivo con un tema de neón, incluyendo:

- Modal animado con bordes brillantes.
- Fondo personalizado con la imagen fondo.png.

¡Cómo Ejecutar el Proyecto?

1. **Preparar el entorno:**
    - Instalar Node.js.
    - Crear una carpeta llamada public y colocar en ella los archivos index.html, styles.css, script.js y la carpeta images con fondo.png.
2. **Instalar las dependencias:**

    npm install

3. **Iniciar el servidor:**

    npm start

4. **Acceder al juego:**

    Abrir un navegador y dirigirse a http://localhost:3000.

Conclusión

El proyecto **"Piedra, Papel o Tijera"** destaca por su diseño intuitivo y funcionalidad en tiempo real. Las herramientas utilizadas, como **Socket.IO**, demuestran su capacidad para manejar comunicación bidireccional de manera eficiente. El diseño visual añade un toque moderno y atractivo, mientras que la estructura modular permite futuras expansiones, como soporte para más jugadores o integración con bases de datos.