OBJETIVO

En esta guía aprenderemos a usar la tecnología websockets, para que a futuro puedas implementarlo en tus proyectos node.

WebSocket

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional entre el servidor y el navegador del usuario. Se entiende por comunicación bidireccional, cuando el servidor puede enviar la información directamente al cliente desde el momento de la conexión.

¿Por qué utilizar Websockets?

Con Websockets los datos que se envían son mas eficientes que en HTTP respecto al tamaño y la velocidad. Los websockets son necesarios cuando debe haber una comunicación constante en el servidor y el cliente. Anteriormente el navegador tenia que consultar constantemente cambios en el servidor cada cierto tiempo lo cual era tedioso y un problema grave de performance en nuestra aplicaciones.

Socket.io

Socket.io es una biblioteca escrita completamente en Javascript que implementa las especificaciones de Websockets en NodeJS, consiste de un Servidor NodeJS y una biblioteca javascript en el lado del cliente.

Características de Socket.IO

Comunicación en Tiempo Real:

Permite a los clientes y servidores intercambiar datos en tiempo real sin la necesidad de recargar la página web.

Comunicación Bidireccional:

Tanto el servidor como el cliente pueden enviar y recibir mensajes, lo que permite una comunicación eficiente y fluida.

Multiprotocolos:

Si WebSockets no están disponibles, Socket.IO puede usar otros protocolos como AJAX long polling, Flash Sockets, etc., asegurando la compatibilidad en la mayoría de los entornos. Eventos:

Usa un modelo basado en eventos, similar al que se usa en Node.js, lo que facilita la programación de la lógica de la aplicación.

Autenticación y Autorización:

Soporta autenticación y autorización de usuarios, lo cual es crucial para muchas aplicaciones. Salas y Espacios de Nombre (Rooms and Namespaces):

Permite la creación de canales lógicos llamados “rooms” y “namespaces” para segmentar la comunicación entre diferentes grupos de clientes.

Cómo Funciona Socket.IO

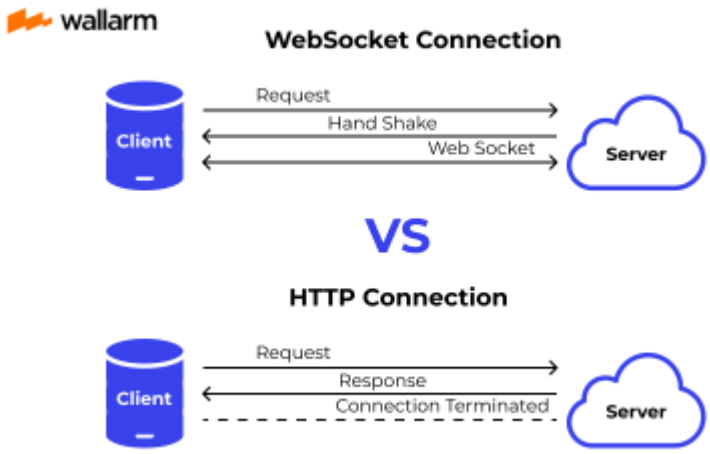
Socket.IO tiene dos componentes principales:

Servidor Socket.IO:

Este componente se ejecuta en el servidor. En un entorno Node.js, se instala como un módulo de Node y se integra con un servidor HTTP.

Cliente Socket.IO:

Este componente se ejecuta en el navegador del usuario. Es una biblioteca de JavaScript que se incluye en la página web del cliente.



En esta guía crearemos una aplicación de chat básico, para implementarlo en nuestro proyecto primeramente debemos instalar las siguientes dependencias:

```
npm install express socket.io
```

En el lado del servidor iniciamos nuestro servidor nuestro web app:

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);

app.get('/', (req, res) => {
```

```
  res.sendFile(__dirname+"/index.html");
});

server.listen(3000, () => {
  console.log('Servidor iniciado en: http://localhost:3000');
});
```

Creamos la interfaz web para representar el cliente (index.html):

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Socket.IO chat</title>
    <style>
      body { margin: 0; padding-bottom: 3rem; font-family: -apple-system,
BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif; }

      #form { background: rgba(0, 0, 0, 0.15); padding: 0.25rem; position: fixed;
bottom: 0; left: 0; right: 0; display: flex; height: 3rem; box-sizing: border-
box; backdrop-filter: blur(10px); }
      #input { border: none; padding: 0 1rem; flex-grow: 1; border-radius: 2rem;
margin: 0.25rem; }
      #input:focus { outline: none; }
      #form > button { background: #333; border: none; padding: 0 1rem; margin:
0.25rem; border-radius: 3px; outline: none; color: #fff; }

      #messages { list-style-type: none; margin: 0; padding: 0; }
      #messages > li { padding: 0.5rem 1rem; }
      #messages > li:nth-child(odd) { background: #efefef; }
    </style>
  </head>
  <body>
   <ul id="mensajes"></ul>
   <form id="form" action="">
    <input id="input" autocomplete="off" /><button>Send</button>
   </form>
  </body>
</html>
```

Integrando socket.io

Para integrar socket.io en nuestro proyecto lo debemos configurar en 2 partes:

- En el lado del servidor debemos enlazarlos al servidor Http de Node.
- En el lado del cliente debemos cargar la libreria cliente socket.io.

Al código anterior de nuestro servidor, agregamos las siguientes líneas:

app.js

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);

const { Server } = require("socket.io");
const io = new Server(server);

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', (socket) => {
  console.log('Usuario Conectado');
});

server.listen(3000, () => {
  console.log('Servidor iniciado en: http://localhost:3000');
});
```

En el lado del cliente (index.html) antes de la etiqueta </body> iniciamos socket.io con la libreria javascript:

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();
</script>
```

Una vez añadida la línea anterior podremos reiniciar el servidor y ver en la consola el mensaje `Nuevo usuario conectado`

Emitir de eventos desde el cliente

La idea principal de Socket.io es que puedas enviar y recibir eventos con cualquier objeto que se pueda codificar en JSON o datos binarios.

Para enviar un al servidor el mensaje que escribe el usuario en el lado del cliente (index.html) invocamos al método `socket.emit` :

```
<script src="/socket.io/socket.io.js"></script>
```

```
<script>
  var socket = io();
 var form = document.getElementById("form");
  var input = document.getElementById("input");
  form.addEventListener('submit',function(e){
    e.preventDefault();//evita que el formulario se envie
    if(input.value){
      socket.emit('Nuevo_mensaje_cliente',input.value);
      input.value = '';
    }
  });
  </script>
```

Emisión de eventos desde el servidor

Desde el lado del servidor podemos enviar eventos con el método `io.emmit`:

app.js

```
io.on("connection",(socket )=>{
    console.log("Nuevo Usuario conectado");
    // socket.broadcast.emit("Hola! Bienvenido al chat");

    socket.on('Nuevo_mensaje_cliente',(msj)=>{
       console.log("Mensaje: "+msj);
       io.emit('Nuevo_mensaje_from_serv',{'texto':msj});//enviando un mensaje a
todos los clientes conectados
    });

    socket.on("disconnect",()=>{
            console.log("usuario desconectado");
        }
    );
});
```

En el ejemplo anterior con el método `socket.on('Nuevo_mensaje_cliente…'` estamos escuchando eventos y con `io.emit('Nuevo_mensaje'…` estamos emitiendo un mensaje a todos los clientes conectados

Escucha de eventos en el cliente

index.html

```
<div id="mensajes"></div>

<script src="/socket.io/socket.io.js"></script>
<script>
  if(socket){
```

```
        socket.on('Nuevo_mensaje_from_serv', (data) => {
            console.log(data);
            dataActual = document.getElementById('mensajes').textContent;
            document.getElementById('mensajes').textContent = dataActual+"\n
"+data.texto;
            console.log(data.mensaje);
        }
        );
    }
</script>
```

Otras alternativas

Existen otras varias alternativas para implementar websockets como por ejemplo:

- Librería nativa `ws`
- Librería `pusher`
- Firebase

Repositorio

El link del repositorio de esta unidad se encuentra en el siguiente enlace:

https://github.com/F1852D160/03-websocketejemplo.git