Websockets

- El uso de **websockets** nos permite la comunicación en tiempo real y bidireccional entre el cliente y el servidor.
- Es común su uso en aquellas aplicaciones que necesitan una interacción multi usuario o en las que necesitemos recuperar en tiempo real ciertos datos que se van actualizando en el servidor (gráficos)
- Hay una diferencia fundamental con las comunicaciones a través del protocolo HTTP
  - En este tipo de comunicaciones se mantiene una conexión entre cliente y servidor pudiendo establecer cierto estado en ls misma.
  - Cuando se abre la conexión se mantiene abierta hasta que uno de los dos cierre el socket.
  - Una vez tengamos abierta la conexión no debemos duplicar la misma porque nos puede dar problemas de rendimiento y de uso de memoria.
- Podemos trabajar con websockets directamente con la API que tenemos definida en el navegador
  - También existen librerías interesantes como Socket.io que nos permiten trabajar de manera más cómoda.
  - El hook **useWebSocket** también nos ofrece una serie de herramientas por encima de la API nativa de websocket.

- Para poder crear una comunicación a través de WS necesitamos un **SERVIDOR NODEJS**

```
const { WebSocketServer } = require("ws")
const http = require("http")
```

```
const uuidv4 = require("uuid").v4
const url = require("url")

const server = http.createServer()
const wsServer = new WebSocketServer({ server })

const port = 8000
const connections = {}
const users = {}
const reports = [];

const handleMessage = (bytes, uuid) => {
    const message = JSON.parse(bytes.toString())
    const user = users[uuid]
    // user.state = message
    reports.push({ user, text: message });
    console.log(reports);
    broadcast()

    console.log(
        `${user.username} ha registrado un nuevo mensaje: ${mess
    )
}

const handleClose = (uuid) => {
    console.log(`${users[uuid].username} disconnected`)
    delete connections[uuid]
    delete users[uuid]
    broadcast()
}

const broadcast = () => {
    Object.keys(connections).forEach((uuid) => {
        const connection = connections[uuid]
        const message = JSON.stringify({ users, reports })
        connection.send(message)
```

```
    })
}

wsServer.on("connection", (connection, request) => {
    const { username } = url.parse(request.url, true).query
    console.log(`${username} connected`)
    const uuid = uuidv4()
    connections[uuid] = connection
    users[uuid] = {
        username
    }
    connection.on("message", (message) => handleMessage(message,
    connection.on("close", () => handleClose(uuid))
})

server.listen(port, () => {
    console.log(`WebSocket server is running on port ${port}`)
})
```

- Dentro de la estructura de nuestra aplicación de React, podríamos detectar los últimos mensajes y enviar a través de **useWebSocket**

```
import { useRef } from "react";
import useWebSocket from "react-use-websocket";

interface Report {
    user: any,
    text: string
}

type WSResponse = { users: any[], reports: Report[] };

const NewReport = () => {
```

```
    const WS_URL = `ws://127.0.0.1:8000`;
    const inputRef = useRef<HTMLInputElement>(null);

    const { sendJsonMessage, lastJsonMessage } = useWebSocket<WS
        queryParams: { username: 'mario' },
        // Mediante esta propiedad, la conexión se comparte a tr
        share: true,
    });

    console.log(lastJsonMessage);

    return <div>
        <h2>Envío de incidencias</h2>
        <div>
            <label>Incidencia</label>
            <input type="text" ref={inputRef} />
            <button onClick={() => sendJsonMessage(inputRef.curr
        </div>
        <div className="reports">
            {lastJsonMessage?.reports.map(report => (
                <div className="report">
                    <p>{report.user.username}: {report.text}</p>
                </div>
            ))}
        </div>
    </div>

}

export default NewReport;
```

SOCKET.IO

- Lo mismo se podría hacer a través de la librería Socket.io
- El **SERVER**

```
const http = require('http');

const server = http.createServer();

const io = require("socket.io")(server,
    { cors: { origin: "*" } }
);

io.on('connection', socket => {
    console.log('Nuevo cliente conectado');

    socket.broadcast.emit('chat_message', {
        usuario: 'INFO',
        texto: 'Se ha conectado un nuevo usuario'
    })

    socket.on('chat_message', (data) => {
        io.emit('chat_message', data);
    });
})

server.listen(3000);
```

- En el **FRONT**

```
import './App.css';
import { io } from 'socket.io-client';
import { useEffect, useState } from 'react';
import { LiMensaje, UlMensajes } from './ui-components';

const socket = io('http://localhost:3000');
```

```
function App() {

  const [isConnected, setIsConnected] = useState(socket.connecte
  const [mensajes, setMensajes] = useState([]);
  const [nuevoMensaje, setNuevoMensaje] = useState('');

  useEffect(() => {
    socket.emit('TEST');

    socket.on('connect', () => {
      setIsConnected(true);
    });

    socket.on('chat_message', data => {
      console.log(data);
      setMensajes(mensajes => [...mensajes, data])
    });

    return () => {
      socket.off('connect');
      socket.off('chat_message');
    };
  }, []);

  const enviaMensaje = (event) => {
    if (event.type === 'keydown' && event.code !== 'Enter') {
      return;
    }
    socket.emit('chat_message', {
      usuario: socket.id,
      texto: nuevoMensaje
    });
  }

  return (
```

```
    <div className="App">
      <h2>Estado: {isConnected ? 'CONECTADO' : 'NO HAY CONEXIÓN
      <h3>ID: {socket.id}</h3>
      <UlMensajes>
        {mensajes.map(mensaje => (
          <LiMensaje>{mensaje.usuario} {mensaje.texto}</LiMensa
        ))}
      </UlMensajes>
      <input
        type="text"
        onChange={e => setNuevoMensaje(e.target.value)}
        onKeyDownCapture={enviaMensaje}
      />
      <button onClick={enviaMensaje}>Enviar</button>
    </div>
  );
}

export default App;
```

```
import styled from 'styled-components';

const UlMensajes = styled.ul`
    max-width: 800px;
    margin: 10px auto;
    list-style: none;
    display: flex;
    flex-direction: column;
    gap: 5px;
`;

const LiMensaje = styled.li`
    background-color: lightblue;
    border: 2px solid dodgerblue;
    padding: 10px 20px;
```

```
`


export {
    UlMensajes, LiMensaje
}
```