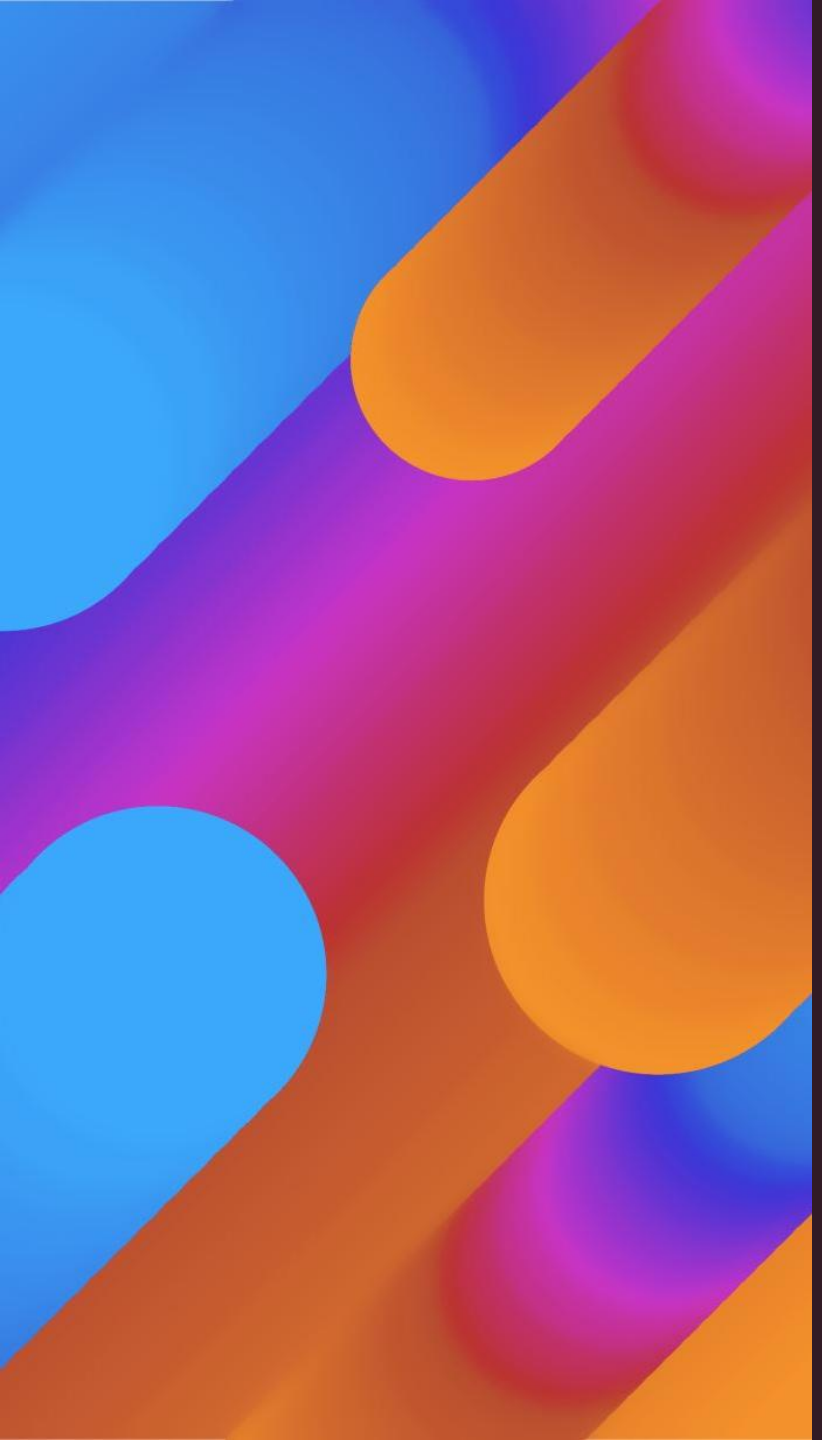
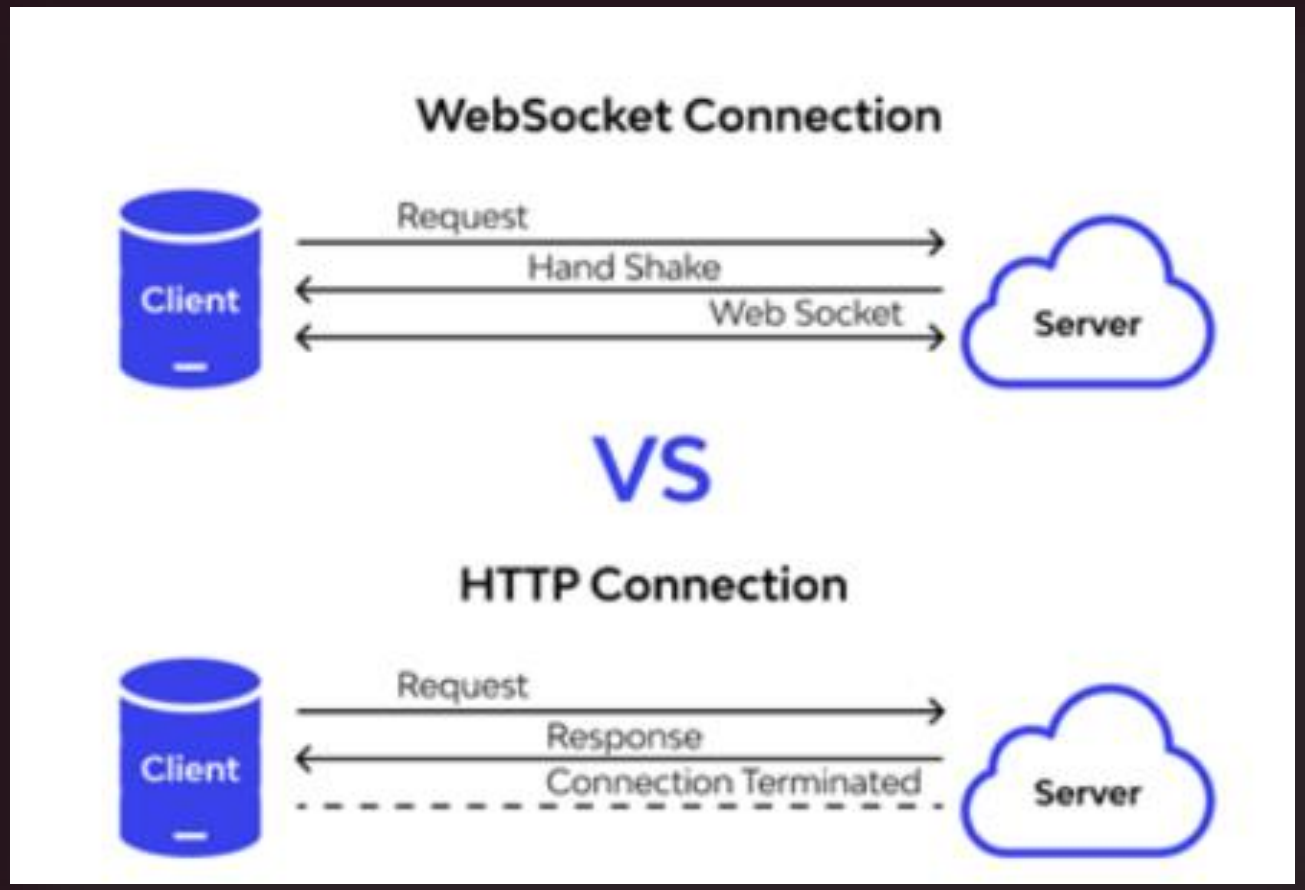


WEBSOCKETS

Clase Particular de Desarrollo de Aplicativos

¿QUÉ SON LOS WEBSOCKETS?

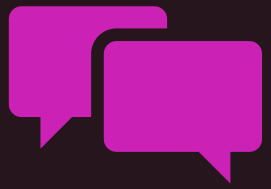
- Los Websockets son un protocolo de comunicación que permite que el navegador y el servidor se **mantengan conectados de forma continua**.
- A diferencia de las conexiones normales (HTTP), los Websockets, **pueden enviarse mensajes mutuamente en cualquier momento** sin volver a conectarse entre sí.



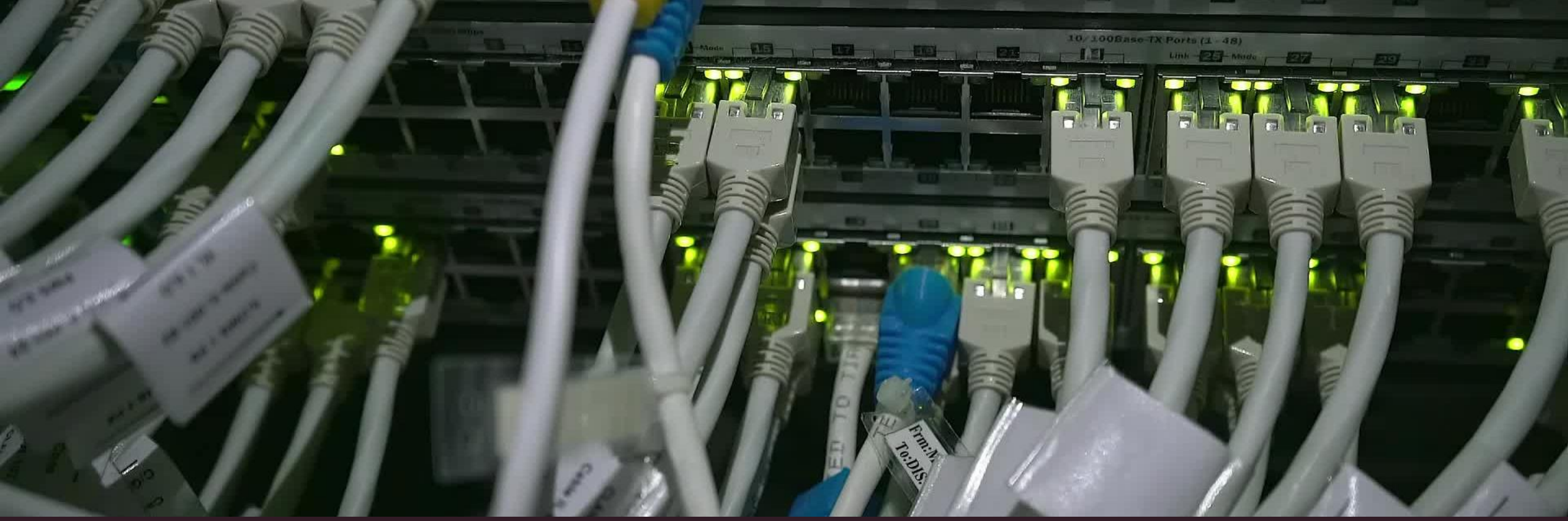
¿PARA QUÉ SE PUEDEN USAR LOS WEBSOCKETS?

Un videojuego multijugador

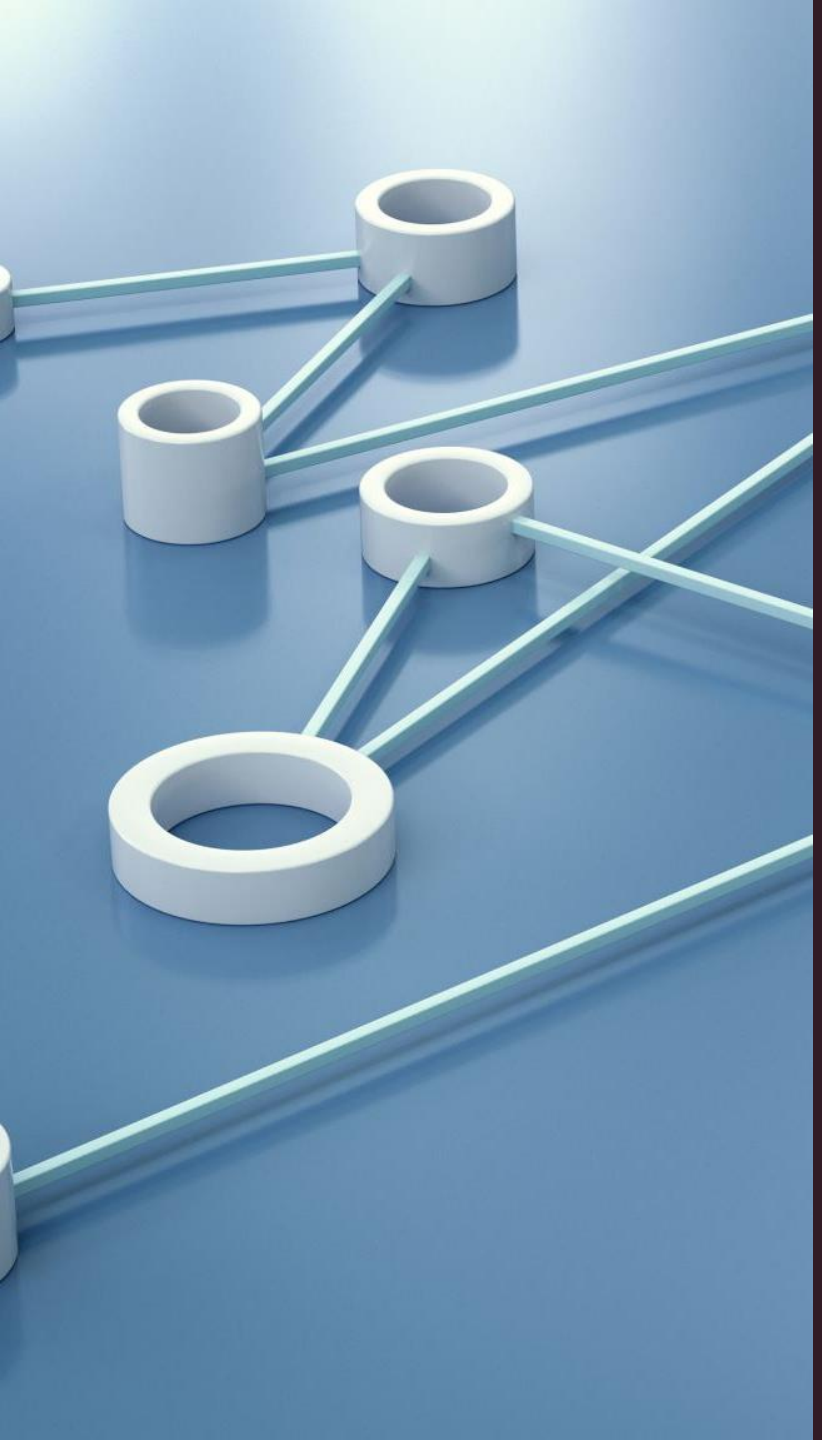
Una notificación en una red social



Una sala de chat para mandar mensajes en tiempo real



SOCKETS EN JAVASCRIPT

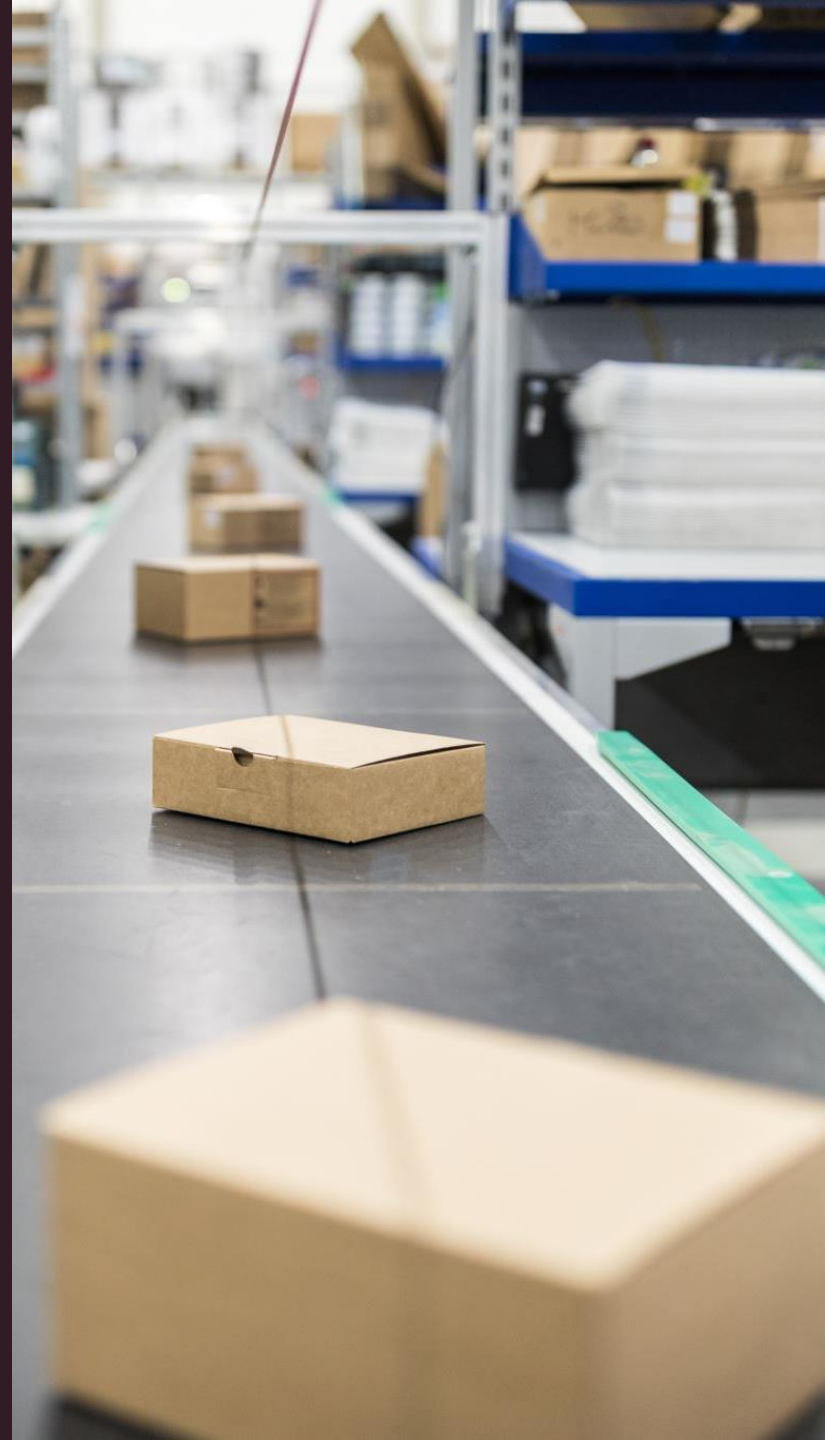


¿QUÉ ES SOCKET.IO?

- Socket.io es un *paquete de Node.js* que permite agregar **comunicación en tiempo real** a las aplicaciones web.
- Establece una conexión **mutua y bidireccional** entre cliente y servidor.
- Nos permite enviar y recibir **mensajes instantáneamente**, sin necesidad de que el navegador recargue la página.
- Los mensajes se envían como **eventos** que se pueden personalizar, como "mensaje nuevo", "pingAll", etc.

INSTALACIÓN DEL PAQUETE WEBSOCKET

- Para el servidor:
  npm install socket.io
- Para el cliente:
  npm install socket.io-client



OBJETO IO Y EL OBJETO SOCKET

Cómo configurar y controlar nuestros sockets

¿QUÉ ES EL OBJETO IO? (SERVIDOR)

- Io es el objeto principal de socket.io en nuestro servidor
- Permite gestionar las conexiones con múltiples clientes a través de sus métodos
- Puede enviar o recibir mensajes de varios clientes simultáneamente
- *Debe ser inicializado y configurado de la siguiente manera:*

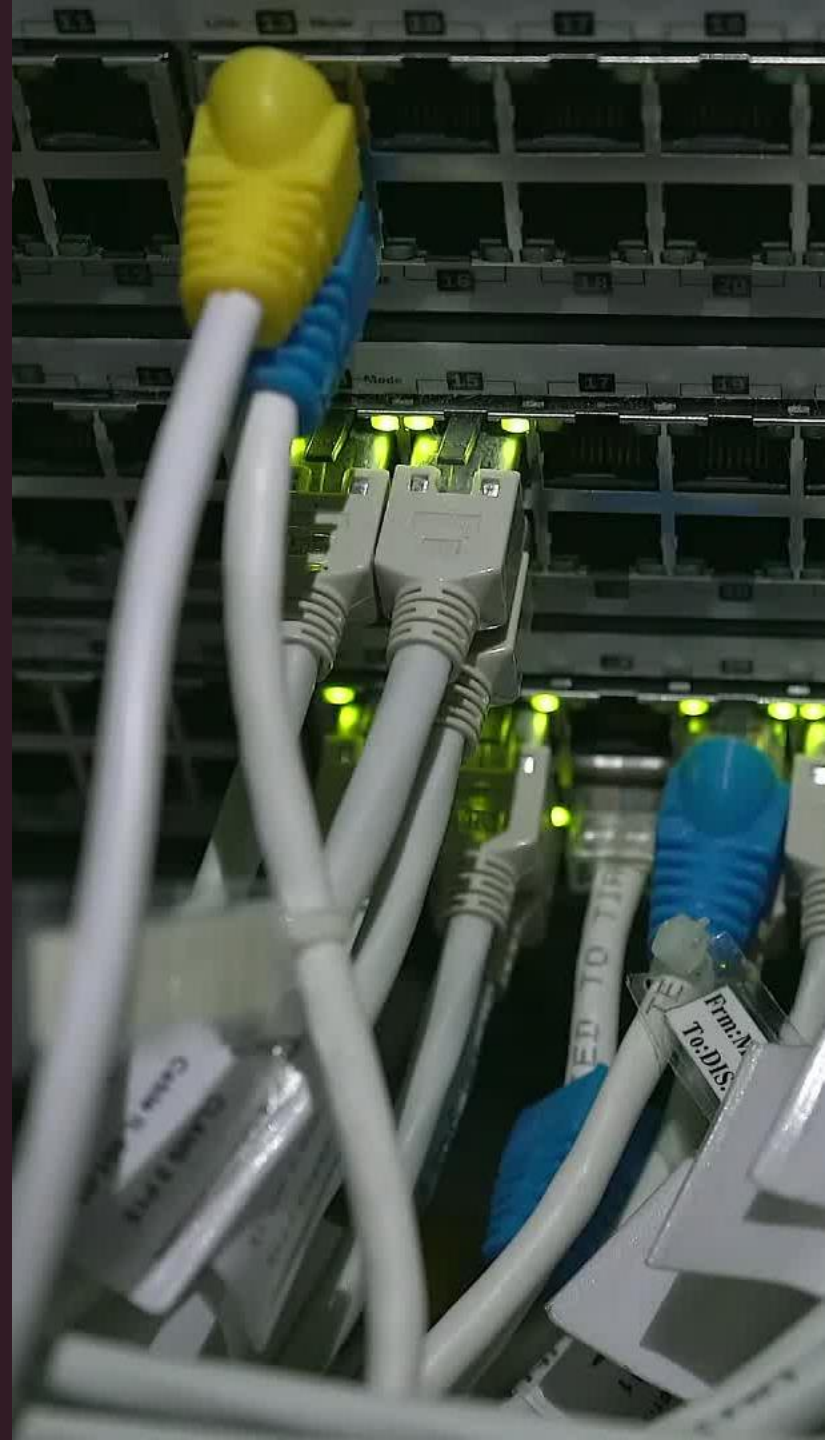
```
const io = require('socket.io')(server, {
    cors: {

        // Permitir los orígenes localhost:3000 y localhost:3001
        origin: ["http://localhost:3000", "http://localhost:3001"],

        methods: ["GET", "POST", "PUT", "DELETE"],  // Métodos permitidos
        credentials: true                           // Habilitar el envío de cookies
    }
});
```

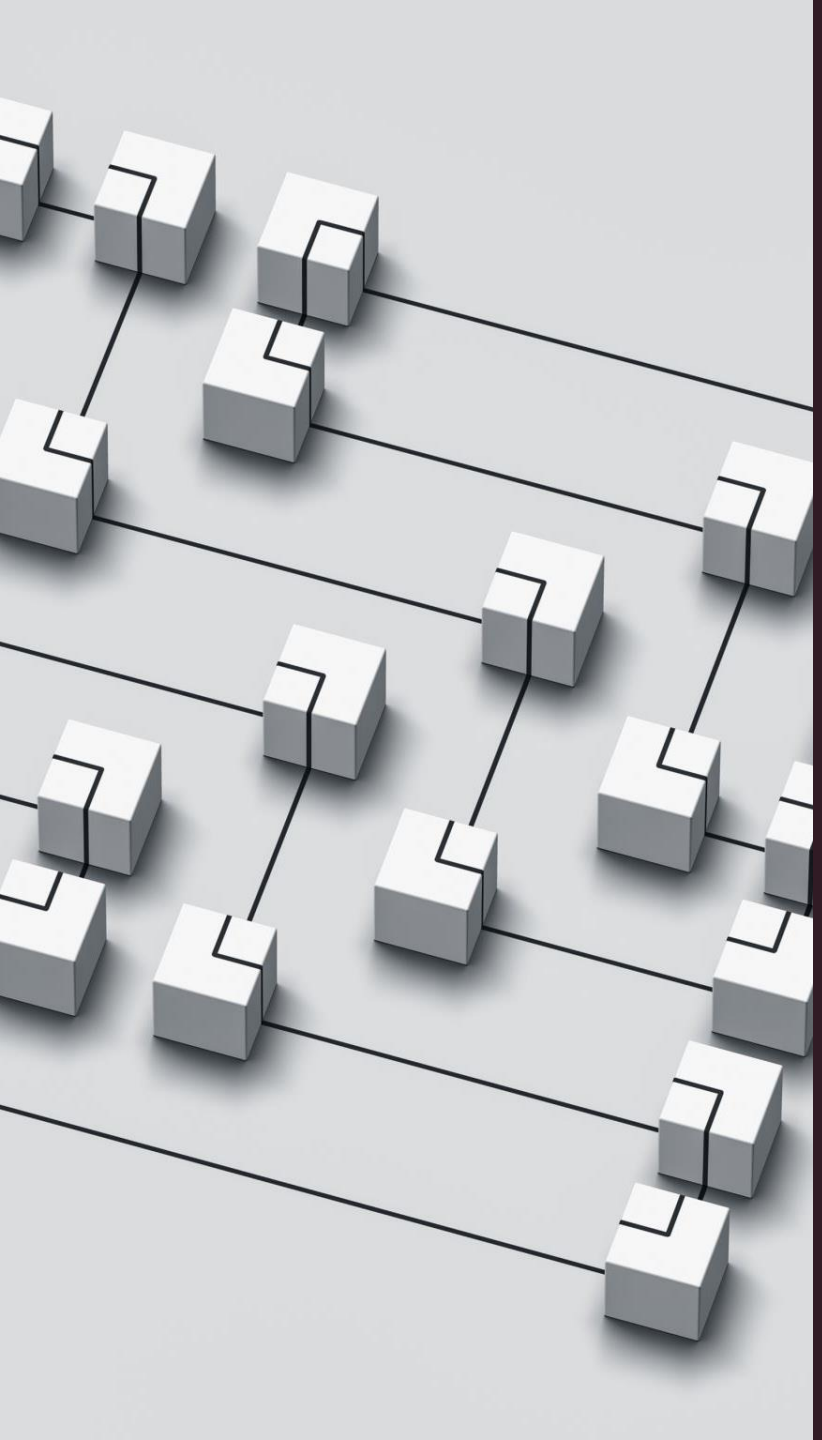
OBJETO SOCKET (CLIENTE Y SERVIDOR)

- Socket es un **objeto** que se crea al momento de establecer una conexión cliente-servidor
- Representa la conexión activa entre un cliente (un cliente por socket) y nuestro servidor.
- El cliente lo usa para comunicarse con el servidor
- El servidor lo usa para comunicarse con un cliente en particular.



MÉTODOS DE LOS OBJETOS IO Y SOCKET

.emit() y .on(), desde el lado del cliente y el lado del servidor



MÉTODO .EMIT()

- Es un método que lo puede usar el objeto socket o el objeto io.
- Se usa para enviar un evento junto con datos al otro lado de la conexión.
- Recibe 2 parámetros:

1. Nombre del Evento (Obligatorio): Un string que representa la "id" del evento.

2. Datos (Opcional): Cualquier tipo de dato que tenga que recibir el otro lado del evento. Generalmente un objeto.

```
io.emit('newMessage', { text: `Hello everybody!` });

socket.emit('pingAll')
```

MÉTODO .ON()

- Es un método que puede ser utilizado por el objeto socket o el objeto lo
- Se usa para "escuchar" un evento en particular que fue emitido desde el otro lado de la conexión.
- Recibe 2 parámetros:
  1. Nombre Del Evento (Obligatorio): Debe coincidir con el nombre emitido
  2. Una función Callback (Obligatorio): Es la función que se va a ejecutar una vez se detecte el evento. *"¿Qué debo hacer cuando detecte este evento?"*

```
socket.on("pingAll", () => {
    console.log("Ping!")
})

socket.on("newMessage", (data) => {
    console.log("Llegó el mensaje", data.text)
})
```

¿DÓNDE SE PONEN LOS .ON Y .EMIT?

BACKEND

- Ambos van dentro del io.on de connection como marca la imagen

```
io.on("connection", (socket) => {
    const req = socket.request;

    // Escribir aquí dentro los emits y las escuchas

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

FRONTEND

- Los socket.on deben ir siempre dentro de un useEffect de un componente sin dependencias.
- Los socket.emit pueden ir en cualquier lugar de un componente.

LISTADO DE EMITS ÚTILES

Tipo de Emisión	Sintaxis	Descripción
Emitir a un solo socket	`socket.emit('evento', data);`	Envía el evento y datos solo al socket específico (cliente actual en `io.on('connection')`).
Emitir a todos los sockets	`io.emit('evento', data);`	Envía el evento y datos a todos los clientes conectados.
Emitir a todos menos uno	`socket.broadcast.emit('evento', data);`	Envía el evento y datos a todos los clientes excepto el socket que emite el mensaje.

EJEMPLO DE SOCKET.ON DENTRO DE UN USEEFFECT

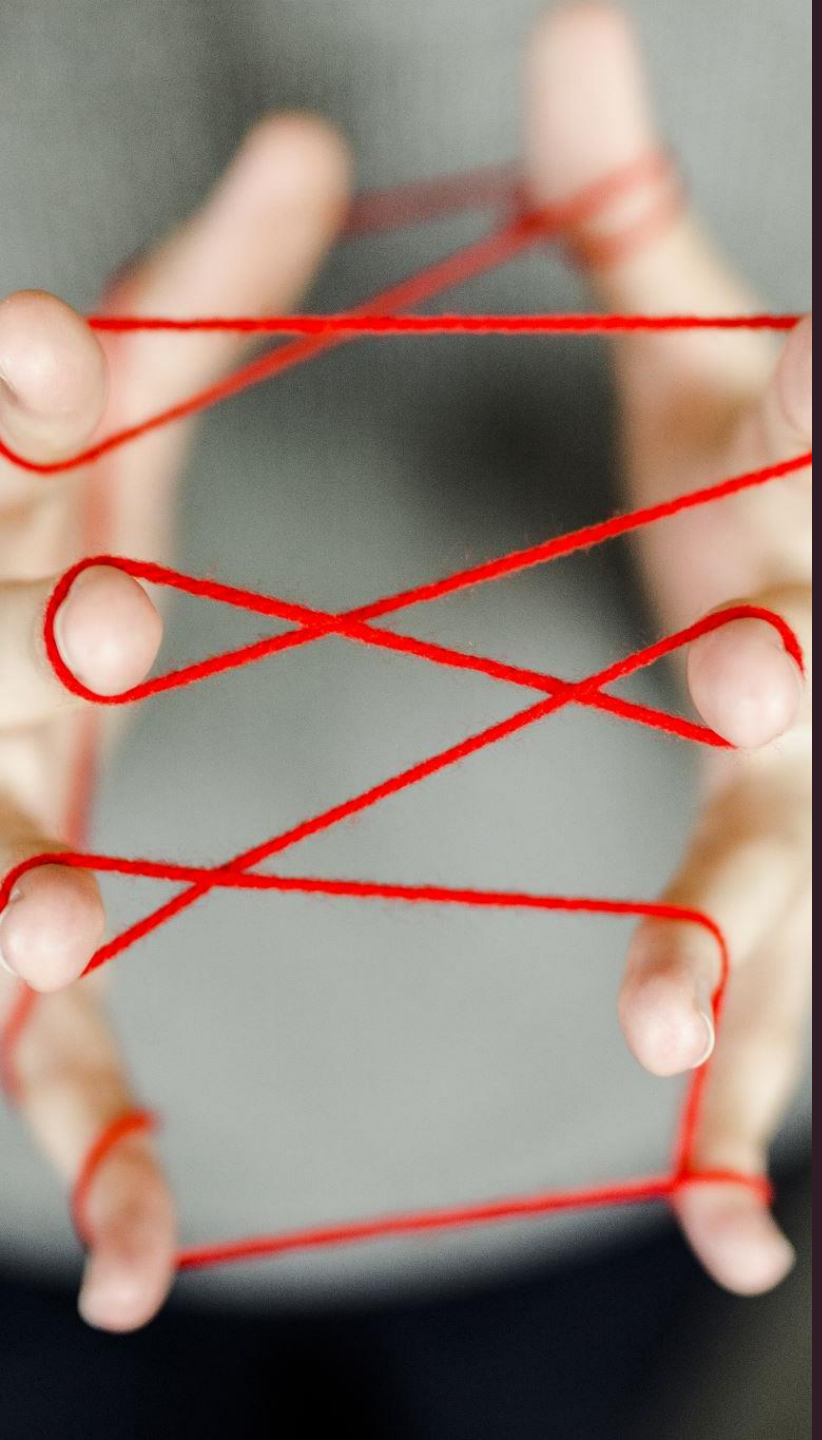
```jsx
import { io } from "socket.io-client"
import { useEffect, useState } from "react"

export default function SocketComponent({ url }) {
    // Para este ejemplo pasamos el url como un prop. Siempre es un string
    const socket = io(url)

    useEffect(() => {
            // Configura un evento al montarse el componente
            socket.on('newMessage', (data) => {
            console.log("Llegó un nuevo mensaje!", data)
            });

            // Limpiar el evento cuando el componente se desmonte
            return () => {
            socket.removeAllListeners('newMessage'); // Se borran todos los eventos
            socket.disconnect(); // Desconectar cuando se desmonta el componente
            };
      },
      [] // Array de dependencias vacío. Los eventos solo se configuran al inicio.
    );

    return ( <> </> )
}
```



INTERVALO DE ACTIVIDAD

Hacer que el backend le envíe un mensaje al frontend cuando un componente se conecta

ROOMS (SALAS)

¿QUÉ ES UNA ROOM?

- Las rooms (salas) en Socket.io son una funcionalidad que permite agrupar a los clientes conectados en diferentes "espacios" dentro del servidor, llamados rooms o salas.
- Se pueden agrupar distintos sockets en una room para enviarles mensajes de forma exclusiva a ellos y nadie más.
- La room de un determinado socket la puede controlar el cliente o el servidor a voluntad con socket.join() y socket.leave() y pasándoles un string (roomName)
- El servidor puede emitir mensajes a ciertas rooms con *io.to(roomName).emit(event, data)*