# REFERENCIA RÁPIDA DE SOCKET.IO

Eventos, salas, espacios de nombres, middleware, patrones en tiempo real

## Configuración

### Configuración del Servidor (Node.js)

```
import { Server } from "socket.io";
const io = new Server(3000, {
  cors: { origin: "http://localhost:5173" }
});
```

### Configuración del Cliente

```
import { io } from "socket.io-client";
const socket = io("http://localhost:3000");
```

### Con Express

```
import express from "express";
import { createServer } from "http";
import { Server } from "socket.io";
const app = express();
const server = createServer(app);
const io = new Server(server);
```

### Opciones del Servidor

| | |
|---|---|
| `cors` | Configuración CORS para origen cruzado |
| `path` | Ruta personalizada (predeterminada: /socket.io) |
| `pingInterval` | Intervalo de latido (ms, predeterminado 25000) |
| `pingTimeout` | Tiempo de espera antes de desconectar (predeterminado 20000) |
| `maxHttpBufferSize` | Tamaño máximo de mensaje en bytes (predeterminado 1MB) |

## Eventos

### Eventos Integrados (Servidor)

| | |
|---|---|
| `connection` | El cliente se conecta |
| `disconnect` | El cliente se desconecta |
| `disconnecting` | El cliente se está desconectando (aún en salas) |
| `error` | Evento de error |

### Eventos Integrados (Cliente)

| | |
|---|---|
| `connect` | Conectado al servidor |
| `disconnect` | Desconectado del servidor |
| `connect_error` | La conexión falló |
| `reconnect` | Reconectado exitosamente |
| `reconnect_attempt` | Intentando reconectar |

### Ciclo de Vida de la Conexión

```
io.on("connection", (socket) => {
  console.log(`connected: ${socket.id}`);
  socket.on("disconnect", (reason) => {
    console.log(`disconnected: ${reason}`);
  });
});
```

## Emisión

### Emisión desde el Servidor

```
socket.emit("hello", { msg: "world" });
socket.emit("data", arg1, arg2);
io.emit("broadcast", data);
```

### Emisión desde el Cliente

```
socket.emit("chat:message", { text });
socket.emit("update", data, (res) => {
  console.log("ack:", res);
});
```

### Patrones de Emisión

| | |
|---|---|
| `socket.emit(ev, data)` | Enviar solo a este socket |
| `io.emit(ev, data)` | Enviar a todos los clientes conectados |
| `socket.broadcast.emit()` | Todos los clientes excepto el remitente |
| `io.to(room).emit()` | Todos los clientes en la sala |
| `socket.to(room).emit()` | Miembros de la sala excepto el remitente |

## Difusión

### Métodos de Difusión

```
io.emit("msg", data);
socket.broadcast.emit("msg", data);
io.to("room1").emit("msg", data);
io.except("room2").emit("msg", data);
```

### Volátil y Comprimido

| | |
|---|---|
| `socket.volatile.emit()` | Descartar si el cliente no está listo (sin buffer) |
| `socket.compress(true).emit()` | Habilitar compresión por mensaje |
| `io.local.emit()` | Difundir solo al servidor local (multi-nodo) |
| `socket.timeout(5000).emit()` | Emitir con tiempo de espera para confirmación |

## Salas

### Operaciones con Salas

```
socket.join("room-1");
socket.join(["room-1", "room-2"]);
socket.leave("room-1");
io.to("room-1").emit("msg", data);
```

### Propiedades de Salas

| | |
|---|---|
| `socket.rooms` | Conjunto de salas en las que está este socket |
| `socket.id` | Cada socket se une automáticamente a su sala de ID propio |
| `io.sockets.adapter.rooms` | Mapa de todas las salas y sus miembros |

### Patrones de Salas

```
socket.on("join:room", (room) => {
  socket.join(room);
  io.to(room).emit("user:joined", socket.id);
});
socket.on("disconnecting", () => {
  for (const room of socket.rooms) {
    socket.to(room).emit("user:left", socket.id);
  }
});
```

## Espacios de Nombres

### Crear Espacios de Nombres

```
const chat = io.of("/chat");
const admin = io.of("/admin");
chat.on("connection", (socket) => {
  chat.emit("user:online", socket.id);
});
```

### Cliente Conectando a Espacio de Nombres

```
const chat = io("http://localhost:3000/chat");
const admin = io("http://localhost:3000/admin");
```

### Espacios de Nombres Dinámicos

```
io.of(/^\/project-\d+$/).on("connection",
  (socket) => {
    const ns = socket.nsp.name;
    console.log(`joined namespace: ${ns}`);
  }
);
```

## Middleware

### Middleware del Servidor

```
io.use((socket, next) => {
  const token = socket.handshake.auth.token;
  if (isValid(token)) return next();
  next(new Error("authentication failed"));
});
```

### Middleware de Espacio de Nombres

```
const admin = io.of("/admin");
admin.use((socket, next) => {
  if (socket.handshake.auth.role === "admin")
    return next();
  next(new Error("not authorized"));
});
```

### Propiedades del Middleware

| | |
|---|---|
| `socket.handshake.auth` | Datos de autenticación enviados desde el cliente |
| `socket.handshake.headers` | Cabeceras HTTP de la solicitud inicial |
| `socket.handshake.query` | Parámetros de consulta de la URL de conexión |
| `socket.data` | Datos arbitrarios adjuntados en el middleware |

## Manejo de Errores

### Errores del Lado del Servidor

```
socket.on("action", (data, callback) => {
  try {
    const result = process(data);
    callback({ status: "ok", data: result });
  } catch (err) {
    callback({ status: "error", msg: err.message });
  }
});
```

### Errores del Lado del Cliente

```
socket.on("connect_error", (err) => {
  console.log("connection error:", err.message);
});
socket.io.on("reconnect_failed", () => {
  console.log("reconnection failed");
});
```

### Opciones de Reconexión del Cliente

| | |
|---|---|
| `reconnection` | Habilitar auto-reconexión (predeterminado true) |
| `reconnectionAttempts` | Intentos máximos (predeterminado Infinity) |
| `reconnectionDelay` | Retraso inicial en ms (predeterminado 1000) |
| `reconnectionDelayMax` | Retraso máximo en ms (predeterminado 5000) |

## Confirmaciones

### Cliente Envía, Servidor Confirma

```
// client
socket.emit("save", data, (response) => {
  console.log("server ack:", response);
});
// server
socket.on("save", (data, callback) => {
  callback({ saved: true, id: 42 });
});
```

### Servidor Envía, Cliente Confirma

```
// server
socket.emit("ping", (response) => {
  console.log("client ack:", response);
});
// client
socket.on("ping", (callback) => {
  callback("pong");
});
```

### Con Tiempo de Espera

```
socket.timeout(5000).emit("save", data,
  (err, response) => {
    if (err) console.log("timeout!");
    else console.log("ack:", response);
  }
);
```

## Patrones Comunes

### Sala de Chat

```
io.on("connection", (socket) => {
  socket.on("chat:join", (room) => {
    socket.join(room);
    socket.to(room).emit("chat:joined",
      socket.id);
  });
  socket.on("chat:message", ({ room, text }) => {
    io.to(room).emit("chat:message", {
      from: socket.id, text
    });
  });
});
```

### Presencia en Línea

```
const users = new Map();
io.on("connection", (socket) => {
  users.set(socket.id, socket.handshake.auth);
  io.emit("users:list", [...users.values()]);
  socket.on("disconnect", () => {
    users.delete(socket.id);
    io.emit("users:list", [...users.values()]);
  });
});
```

### Limitación de Velocidad

```
io.use((socket, next) => {
  const ip = socket.handshake.address;
  if (rateLimiter.consume(ip)) return next();
  next(new Error("rate limit exceeded"));
});
```