

# Riferimento rapido Socket.IO

Events, room, namespace, middleware, pattern real-time

## Configurazione

### Configurazione server (Node.js)

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

### Configurazione client

```
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);
```

### Opzioni server

<b>cors</b>	Configurazione CORS per richieste cross-origin
<b>path</b>	Percorso personalizzato (default: /socket.io)
<b>pingInterval</b>	Intervallo heartbeat (ms, default 25000)
<b>pingTimeout</b>	Timeout prima della disconnessione (default 20000)
<b>maxHttpBufferSize</b>	Dimensione massima messaggio in byte (default 1MB)

## Events

### Event integrati (server)

<b>connection</b>	Il client si connette
<b>disconnect</b>	Il client si disconnette
<b>disconnecting</b>	Il client sta disconnettendosi (ancora nelle room)
<b>error</b>	Evento di errore

### Event integrati (client)

<b>connect</b>	Connesso al server
<b>disconnect</b>	Disconnesso dal server
<b>connect_error</b>	Connessione fallita
<b>reconnect</b>	Riconnessione riuscita
<b>reconnect_attempt</b>	Tentativo di riconnessione in corso

### Ciclo di vita della connessione

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

## Emissione

### Emissione dal server

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

### Emissione dal client

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

## Pattern di emissione

<b>socket.emit(ev, data)</b>	Invia solo a questo socket
<b>io.emit(ev, data)</b>	Invia a tutti i client connessi
<b>socket.broadcast.emit()</b>	A tutti i client tranne il mittente
<b>io.to(room).emit()</b>	A tutti i client nella room
<b>socket.to(room).emit()</b>	Ai membri della room tranne il mittente

## Broadcasting

### Metodi di broadcast

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

## Volatile e compresso

<b>socket.volatile.emit()</b>	Scarta se il client non è pronto (nessun buffer)
<b>socket.compress(true).emit()</b>	Abilita la compressione per messaggio
<b>io.local.emit()</b>	Broadcast solo al server locale (multi-nodo)
<b>socket.timeout(5000).emit()</b>	Emissione con timeout per acknowledgement

## Room

### Operazioni sulle room

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

### Proprietà delle room

<b>socket.rooms</b>	Set di room in cui si trova questo socket
<b>socket.id</b>	Ogni socket si unisce automaticamente alla sua room ID
<b>io.sockets.adapter.rooms</b>	Mappa di tutte le room e dei loro membri

### Pattern con room

```
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);
  }
});
```

## Namespace

### Creazione di namespace

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

### Client che si connette a un namespace

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

## Namespace dinamici

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

## Middleware

### Middleware del server

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

### Middleware del namespace

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

### Proprietà del middleware

<b>socket.handshake.auth</b>	Dati di autenticazione inviati dal client
<b>socket.handshake.headers</b>	Header HTTP dalla richiesta iniziale
<b>socket.handshake.query</b>	Parametri query dall'URL di connessione
<b>socket.data</b>	Dati arbitrari allegati nel middleware

## Gestione degli errori

### Errori lato server

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

### Errori lato client

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

### Opzioni di riconnessione del client

<b>reconnection</b>	Abilita la riconnessione automatica (default true)
<b>reconnectionAttempts</b>	Numero massimo di tentativi (default Infinity)
<b>reconnectionDelay</b>	Ritardo iniziale in ms (default 1000)
<b>reconnectionDelayMax</b>	Ritardo massimo in ms (default 5000)

# Riferimento rapido Socket.IO

## Acknowledgement

### Il client invia, il server risponde

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

### Il server invia, il client risponde

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

## Con timeout

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

## Pattern comuni

### Stanza 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
    });
  });
});
```

### Presenza online

```
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()]);
  });
});
```

### Limitazione della velocità

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