

Socket.IO Quick Reference

Events, rooms, namespaces, middleware, real-time patterns

Setup

Server Setup (Node.js)

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

Client Setup

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

With 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);
```

Server Options

cors	CORS configuration for cross-origin
path	Custom path (default: /socket.io)
pingInterval	Heartbeat interval (ms, default 25000)
pingTimeout	Timeout before disconnect (default 20000)
maxHttpBufferSize	Max message size in bytes (default 1MB)

Events

Built-in Events (Server)

connection	Client connects
disconnect	Client disconnects
disconnecting	Client is disconnecting (still in rooms)
error	Error event

Built-in Events (Client)

connect	Connected to server
disconnect	Disconnected from server
connect_error	Connection failed
reconnect	Successfully reconnected
reconnect_attempt	Attempting to reconnect

Connection Lifecycle

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

Emitting

Server Emit

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

Client Emit

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

Emit Patterns

socket.emit(ev, data)	Send to this socket only
io.emit(ev, data)	Send to all connected clients
socket.broadcast.emit()	All clients except sender
io.to(room).emit()	All clients in room
socket.to(room).emit()	Room members except sender

Broadcasting

Broadcast Methods

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

Volatile & Compressed

socket.volatile.emit()	Drop if client not ready (no buffering)
socket.compress(true).emit()	Enable per-message compression
io.local.emit()	Broadcast to local server only (multi-node)
socket.timeout(5000).emit()	Emit with timeout for acknowledgement

Rooms

Room Operations

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

Room Properties

socket.rooms	Set of rooms this socket is in
socket.id	Each socket auto-joins its own ID room
io.sockets.adapter.rooms	Map of all rooms and their members

Room Patterns

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

Namespaces

Creating Namespaces

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

Client Connecting to Namespace

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

Dynamic Namespaces

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

Middleware

Server Middleware

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

Namespace Middleware

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

Middleware Properties

socket.handshake.auth	Auth data sent from client
socket.handshake.headers	HTTP headers from initial request
socket.handshake.query	Query parameters from connection URL
socket.data	Arbitrary data attached in middleware

Error Handling

Server-Side Errors

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

Client-Side Errors

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

Client Reconnection Options

reconnection	Enable auto-reconnect (default true)
reconnectionAttempts	Max attempts (default Infinity)
reconnectionDelay	Initial delay ms (default 1000)
reconnectionDelayMax	Max delay ms (default 5000)

Acknowledgements

Client Sends, Server Acks

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

Socket.IO Quick Reference

Server Sends, Client Acks

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

With Timeout

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

Common Patterns

Chat Room

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

Online Presence

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

Rate Limiting

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