To use socket.io

<npm i socket.io > That should be run in server

<npm i socket.io-client> That should be run in client folder.

**🔹 Line 1**

js

CopyEdit

const express = require('express')

* You're importing the **Express** library.
* Express helps you create a web server (like a website backend).
* After this, you’ll be able to use express() to make a server.

**🔹 Line 2**

js

CopyEdit

const { Server } = require('socket.io')

* This brings in the **Socket.IO Server** class.
* Socket.IO lets your server talk to users instantly (real-time), like in a chat app.

**🔹 Line 3**

js

CopyEdit

const http = require('http')

* This imports Node.js’s built-in http module.
* It's used to actually create a server that can listen to requests (from browsers or other clients).
* Think of it as the foundation your app is built on.

**🔹 Line 4**

js

CopyEdit

const app = express()

* This creates an **Express app**.
* You’ll use app to handle routes like GET, POST, etc.
* Even if you don’t use routes right now, this line prepares your server for them.

**🔹 Line 6**

js

CopyEdit

const server = http.createServer(app)

* This creates an **HTTP server** using your Express app.
* Socket.IO needs this because it doesn’t work with app.listen() — it needs direct access to the raw server.

**🔹 Line 8**

js

CopyEdit

const io = new Server(server, {

cors: {

origin: process.env.FRONTEND\_URL,

credentials: true

}

})

* This creates a new **Socket.IO server** (io).
* It’s connected to your HTTP server (so now, it can handle WebSocket connections).
* cors: This part lets your **frontend** (your client app, like React) connect to this server.
  + origin: Only allow connections from a specific URL (from your .env file).
  + credentials: true: Allows sending cookies or headers with the connection.

**🔹 Line 13**

js

CopyEdit

io.on('connection', (socket) => {

* This runs **every time a user connects** to your Socket.IO server.
* You get a socket object — it represents that single user.
* From here, you can send or receive messages **just with that user**.

**🔹 Line 14**

js

CopyEdit

console.log("connected user", socket.id)

* This prints the user's unique **socket ID** to the console.
* Each user gets a different ID when they connect.

**🔹 Line 16**

js

CopyEdit

socket.on('disconnect', () => {

* This listens for when that **specific user disconnects** (closes tab, loses connection, etc.).
* It’s the opposite of connection.

**🔹 Line 17**

js

CopyEdit

console.log("Disconnected user", socket.id)

* When a user disconnects, this will log their socket ID.

**🔹 Line 20**

js

CopyEdit

module.exports = {

app,

server

}

* This allows you to **use app and server in another file**.
* For example, in another file you might do:

js

CopyEdit

const { app, server } = require('./socket')

server.listen(3000)

* That way, you keep your code organized.

**🧠 Summary (In Plain English)**

* You made an Express app.
* You created a server that also supports real-time connections using Socket.IO.
* You’re listening for new users (connection) and when they leave (disconnect).
* You’re allowing only certain frontends to connect (via cors).
* And finally, you exported the server so you can start it in another file.

If you want, I can show you how to use this with a frontend (like a React app) or how to send and receive custom messages (like chat messages). Would that help?

In **Socket.IO**, a "room" is like a **private group** that sockets can join. You can then send messages **only** to the sockets in that room instead of broadcasting to everyone.

A Set is a special type of **JavaScript object** that lets you store **unique values** — meaning no duplicates allowed.

In **Socket.IO**, socket represents the **individual real-time connection** between your **server** and **one specific client** (usually a browser).

| **Property** | **Purpose** |
| --- | --- |
| socket.id | A unique ID for that connection |
| socket.handshake | Metadata sent during connection (like token) |
| socket.emit(event, data) | Send data to *that one client* |
| socket.on(event, callback) | Listen to events sent *by that client* |
| socket.join(roomName) | Join them to a room |
| socket.disconnect() | Kick them out / end the connection |