Socket.io

The `socket.io` and `socket.io-client` packages are fantastic tools that allow you to easily set up web sockets. We'll be doing a simple example for classwork, but the below at least gives you an idea of how to implement it. As mentioned previously, sockets (and this package) function off of an emitter and listener pattern.

socket.io

The socket.io package is how you would handle sockets server side. Usually, this configuration would be in it's own .conf file but it can also be directly in the server.js file if it is simple enough. As per the docs, an incredibly simple usage could be:

```
import express from 'express';
import { createServer } from "http";
import { Server } from "socket.io";
const app = express();
const server = createServer(app); // Creates an HTTP server
const io = new Server(server, {
    cors: {
        origin: "http://localhost:5173",
        methods: ["GET", "POST"]
    }
});

app.use(cors()); // Allow frontend to connect to the server
app.use(express.json()); // Middleware for JSON requests
io.on("connection", () => {
  /* … */
});
server.listen(8080);
```

The only difference from the usual is that you're implementing the `createServer` function. This is how you expose the server to sockets. With that in mind, let's take a look at how to actually handle the sockets. The `io.on('connection', () => { /* … */ });` is where all of your logic would go. In this example it's paying attention to anytime ANYONE connects to the server. In this instance, both `connection` and `disconnect` are pre-built events. Inside the callback function you can handle emitting events to everyone, or pass it an argument and emit data to only specific people. Let's take a look at both:

```
// socket is the individual connection that was just made
io.on("connection", (socket) => {
  // This is how you would handle if a specific connection sent something such as
a chat message
  socket.on("chat message", () => {
    // send the chat message
  });
```

```
  // This would emit that data to any current connection
  io.emit("event name", data);

  // This would emit that data to ONLY this specific connection
  socket.emit("event name", data);

  // This allows for some sort of functionality when a socket closes because the
user leaves / chooses to disconnect / some other logic.
  // It's a good way to do cleanup functions or show other users that someone has
left
  socket.on("disconnect", () => {});
});

// This would be if you wanted something to happen whenever ANYONE did something.
io.on("customEvent", () => {});
```

Something important to keep in mind is that `io.emit` broadcasts an event to **everyone** connected to the server, whereas `socket.emit` sends it to a specific person (via their own connected socket).

socket.io-client

Basics

The socket.io-client package is what allows you to make connections to the server that has sockets activated. The implementation in standard JavaScript is very straightforward and in fact it can be just as easy in React. As per the docs, a standard implementation could be:

```
import { io } from "socket.io-client";

const socket = io("http://localhost:3006");
socket.on("connect", function () {});
socket.on("event", function (data) {});
socket.on("disconnect", function () {});
```

In the example above, the socket is connecting to the appropriate server backend and then listening for any events. You'll notice that the `event` event has an argument called `data` in its callback function. Like above, that is how you would handle any data being emitted with the event. That data could be anything you choose based off the event you're using. From there if you want to emit something to the server side it would simply be:

```
socketRef.emit("event name", dataToEmit);
```

It really is that straightforward. There are of course more things you can do such as choose a room to connect to, ignore specific events, emit events to specific people or anything in between!

socket.io-client In React

The above can also be used in React. While it isn't considered a good programming pattern, you could very easily just take that code and put it into your needed component and be done with it. However, that causes a different problem as it might cause re-renders depending on where you put it. The way you would ultimately want to handle sockets client side is with some sort of custom hook. That might look something akin to:

```
import { useEffect, useState } from "react";

const useSocketHook = (socket) => {
  const [stateItem, setStateItem] = useState([]);

  useEffect(() => {
    socket.on("someEvent", (data) => {
      setStateItem(data);
    });

    return () => {
      socket.disconnect();
    };
  }, [socket]);

  const sendData = (dataToEmit) => {
    socket.emit("Event", dataToEmit);
  };

  return { stateItem, sendData };
};

export default useSocketHook;
```

You'll notice with the above you're needing to pass in a socket that is pre-connected. This can be bypassed if you have some sort of other argument. If you're not careful though with the `useEffect` hook, you could inadvertently have the hook continuously re-run and re-connect meaning you could never actually handle events properly. There are of course React specific packages and I suggest you look into them to potentially clean up your code further.