Saturday, 17 August 2024

Web Sockets

What is WebSocket?

WebSocket is a communication protocol that provides full-duplex communication channels over a single TCP connection. Unlike HTTP, which follows a request-response model, WebSocket allows for persistent connections where both the client and the server can send and receive messages at any time. This makes WebSocket ideal for real-time applications such as chat apps, online gaming, live notifications, etc.

Why Do We Need WebSocket?

Traditional HTTP connections are not suitable for real-time communication because they rely on frequent polling or long polling, which can be inefficient and resource-intensive. WebSocket addresses this limitation by allowing an open, continuous connection, enabling real-time communication with minimal overhead.

How web socket works ?

Web socket works on the event. The socket server accepts connection from multiple client. Once a client get connected to socket server then server as well as client can communicate with each using using events.

- When server wants to send any message to a client, it will emit the event that client is listening for
- When client wants to send any message to server it will emit the event that server is listening for

Event: Event is consist of two thing. First a string that is event name another an object that is knows as event payload. Whenever any client or server wants to emit an event they have to provide the name of event and the payload (data to be sent by this event). Event name can be anything it is not fixed.

Some predefined events:

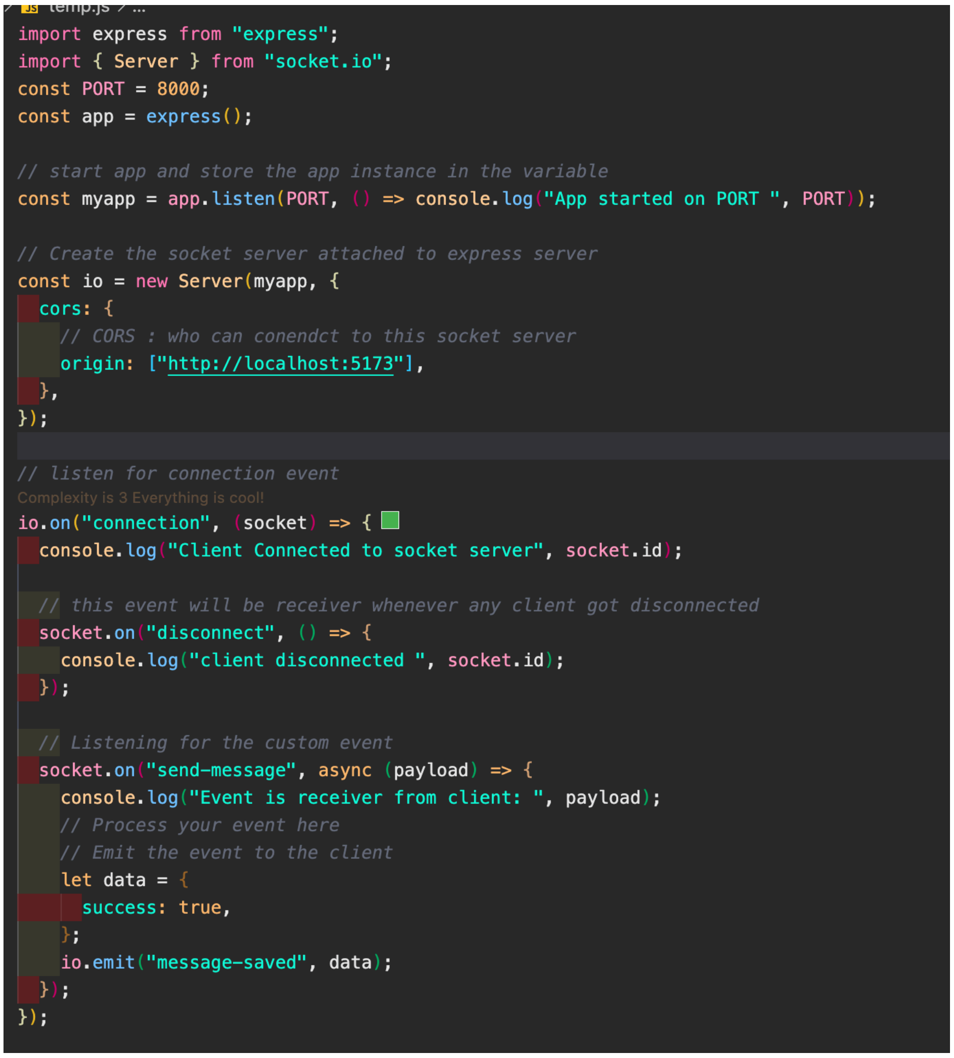
1. *Connection*: This is the event that is triggered whenever any client will connect to the server. So server will listen for the incoming successful connections on the “connection” event
2. *Disconnect*: This event is triggered whenever any client got disconnected from the socket server.

What is on and emit ?

On : It is the method that is used to set listener for a specific event. For example any client or server want’s to listen for any event then they will use the on method. On takes 2 arguments, first the event name and second a callback function to be Called whenever this event will be receiver and the payload of event is passed to your callback function as a parameter.

Emit: Whenever any client or server want’s to emit and event they will use the emit method. Emit will also take 2 arguments. First the event name and second is payload.

How to create a socket server with express:



```
import express from "express";
import { Server } from "socket.io";
const PORT = 8000;
const app = express();

// start app and store the app instance in the variable
const myapp = app.listen(PORT, () => console.log("App started on PORT ", PORT));

// Create the socket server attached to express server
const io = new Server(myapp, {
  cors: {
    // CORS : who can conendct to this socket server
    origin: ["http://localhost:5173"],
  },
});

// listen for connection event
Complexity is 3 Everything is cool!
io.on("connection", (socket) => {
  console.log("Client Connected to socket server", socket.id);

  // this event will be receiver whenever any client got disconnected
  socket.on("disconnect", () => {
    console.log("client disconnected ", socket.id);
  });

  // Listening for the custom event
  socket.on("send-message", async (payload) => {
    console.log("Event is receiver from client: ", payload);
    // Process your event here
    // Emit the event to the client
    let data = {
      success: true,
    };
    io.emit("message-saved", data);
  });
});
```

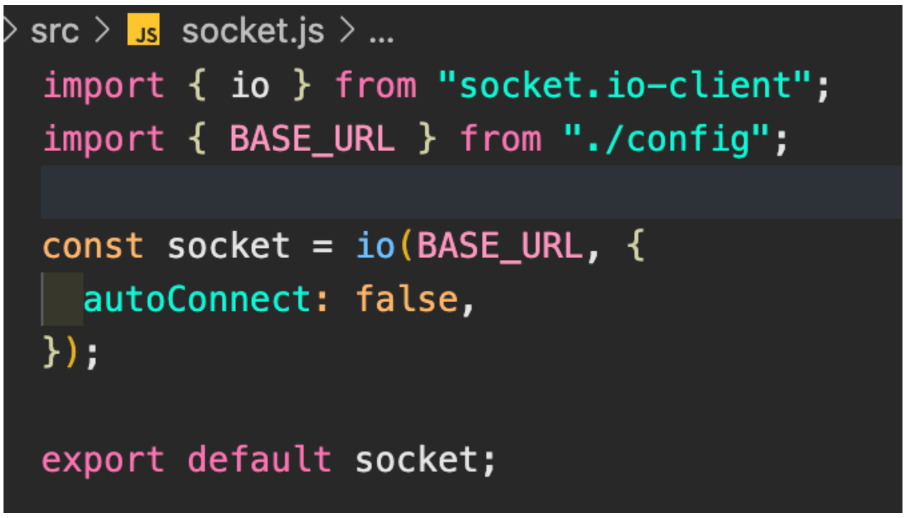
This is just a normal example of how to create a socket server and listen for event and once that event is received how you can emit some other event.

"In our final HMS project we learned it in detailed way so refer to that codebase".

How to connect the client:

Create a socket.js file where you can connect to your socket url. The socket url in our case will be same as express server as socket server is attached to express server.

Following code will just create a reference to the connection that is not active for now as autoConnect is false. So we have to forcefully connect later.

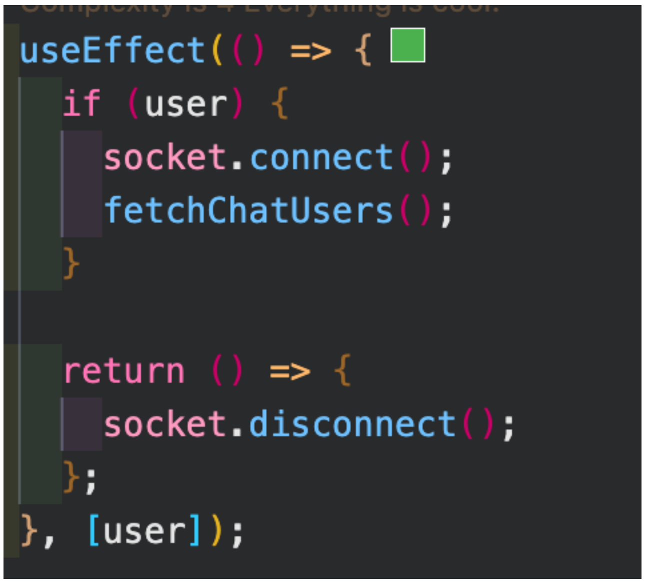


```
src > JS socket.js > ...
import { io } from "socket.io-client";
import { BASE_URL } from "./config";

const socket = io(BASE_URL, {
  autoConnect: false,
});

export default socket;
```

Let's take example if the user logged in then only we will connect to socket server. Here is following example.



```
useEffect(() => {
  if (user) {
    socket.connect();
    fetchChatUsers();
  }

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