NAMASTE NODE.JS SEASON 3

Episode-8

Hand Written Notes

-By Shanmuga Priya

www.linkedin.com/in/shanmuga-priya-e-tech2

Episode-8

web socket and Socket.io

what is Socket.io?

Socket.io is a library that enables low-latency (fast, smooth), bidirectional (client ⇄ server) and event-based communication between a client and a server.

what is webSocket?

→ web socket is a communication protocol that enables full-duplex (communication occurs simultaneously, in both directions), bidirectional communication between a client and a server over a single, persistent connection which allows real-time data exchange.

→ web socket connections are established over the standard TCP/IP Protocal, typically on port 80 (HTTP) (or) 443 (HTTPS).

Steps involved in implementing chat feature:

Step 1: Building UI:

→ Add a button to each connections with the content "chat" on clicking it it should take to the chat Page along with the selected person's user id.

→ Build a new chat component to display msgs and also a input box and button to send new msgs.

Step 2: Implementing server side Connection

Step 1: Install socket.io

```
npm i socket.io
```

Step 2: Configuration of socket.io in app.js (or) new file & include it in app.js

→ earlier we created a server using express but now for configuration of socket.io we need to create it using "http" module (built-in module)

eg:
```
Const http = require ("http").
Const socket = require ("socket.io")
//Create server using http.
   Const server = http.createServer (app)   → we pass the express app
//Socket configuration
   Const io = socket (server, {
                  cors: {
                     origin: frontend url,
                  }
   })
// listening for the events
io.on("connection", (socket) => {
   // handle events
   socket.on ("joinchat", () => { })
})
```

(1)
```
// Replace the express server with this http server.
connectDB().then(()=>{
      server.listen (8000, ()=> { console.log("server connected")
})
```

Step 3: Configuration of socket.io in frontend.

→ As its bidirectional we need to configure socket in both frontend & Backend. for that install a package

```
npm i socket.io-client
```

→ configure this socket.io in a seperate file in utils folder.

```
//socket.js
import io from "socket.io-client"
import {BaseURL} from "./constants"

export const createSocketConnection = () => {
    return io(BaseURL)          → this configuration only works in
                                  development not in production.
                L> Backendurl
}
```

Step4: update the code in chat component

→ In chat component as soon as the page loads we need to establish a connection for that we use useEffect.

```
eg: // chat.jsx .

useEffect(() => {                    ↱ fn created in socket.js in above step
    const socket = createSocketConnection()
// socket connection is made & join chat event is emitted
    socket.emit("joinchat", {userId, targetuserId})   → data that we need to send to backend.
                ↳ this event should match the backend events

// disconnect socket whenever necessary (-X)(-X)
    return () => {
        socket.disconnect()
    }
}, [])
```

Step5: Handling the events in the backend

Step 1) Creating room for joining chat

we need to create a seperate room in a server for each chats where both can communicate.

→ Each room should have a unique roomId and when to 2 person is chatting lets say x & y, the chat roomId of x→y and y→x should be same.

eg: // app.js

```
io.on("connection", (socket) => {
  // handling join the room event
  socket.on("joinchat", ({ firstName, userId, targetUserId }) => {
                          ↳ these are the data passed from the frontend
    const roomId = [userId, targetUserId].sort().join("-")
                   ↓ we are sorting it in order to avoid creating
    2 seperate room for same users (ie x→y & y→x)

    socket.join(roomId)
      ↳ is a method used to join the room.
  })

  // handling the event for sending Message.
  socket.on("sendMessage", ({ firstName, userId, targetUserId, msg }) =>
                           ↓ data from frontend.
  {
    const roomId = [userId, targetUserId].sort().join("-")

    io.to(roomId).emit("messageReceived", { firstName, text })
      ↳ sending a message to a room
                                  ↳ once msg received we are emitting an event we need to listen this up in frontend.
  })
})
```

Step 6: Updating the frontend to send message to backend

→ Creating a state var to store a content from the input box and attach a handler fn to send button to send a msg to backend.

eg: const [msg, setMsg] = useState("")

```
//handler fn to send a msg
const sendMessage = () => {
    Const socket = CreateSocketConnection()
    socket.emit ("sendMessage", {
        firstName, userId, targetUserId, msg
    })
}.

//attach it to btn & input box
<input type="text" value={msg} onchange={(e) => setmsg(e.target.value)} />

<button onclick={sendMessage}>Send</button>.
```

Step 7: Listening up the "Message Received" event emitted from Backend in frontend.

→ create a new state var to store the msgs to display it in UI.

→ whenever the event is emitted from a BE we listen it in F.E and push that message to the state var.

```
eg: const [messages, setMessages] = useState([])

//Inside useEffect
socket.on("messageReceived", ({firstName, text}) => {
    setMessages((messages) => [...messages, {firstName, text}])
})
```

→ Loop through this state var & display it in UI.

Step 8: Make the RoomId encrypted in Backend

→ Instead of using the userId & targetuserId as if we can hash it for better security.

eg:
```
const crypto = require("crypto")

// fn to hash Id's
Const getSecretRoomId = (userId, targetUserId) => {
    return crypto.createHash("sha256")
        .update([userId, targetuserId].sort().join("_"))
        .digest("hex")
}

// use it to generate roomId.
socket.on("joinchat", ({ firstName, userId, targetuserId }) => {
    const roomId = getSecretRoomId(userId, targetuserId)

    socket.join(roomId)
})
```

Homework:

Refer the authentication in socket.io documentation & implement it in the project.

---

x