NAMASTE NODE.JS SEASON 3

Episode-9
Hand Written Notes

-By Shanmuga Priya

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

Episode-9

Building Real time Live chat feature

-> It is the continuation of the episode-8 , where we have just stored the messages in the state var. The problem with that is whenever we refresh the Page the chats also get erased. In this episode we will learn how to store that chat in DB and retrieve it.

Step 1: Create a Message Model in B-E

```
const  mongoose = require ("mongoose")

const messageschema = new mongoose.Schema(
    { senderId: {
          type: mongoose.schema.Types.ObjectId,
          ref: "User",
          required: true,
      },
      text: { type: string, required: true },
    },
    { timestamps: true }
)

const ChatSchema = new mongoose.schema({
senders&      Participants: [ { type: moongoose.Schema.Types.ObjectId,
receiverId                      ref, "User", required: true } ],

      messages: [ message schema ]  -> messageschema defined above
})

const Chat = mongoose.model ("Chat", chatschema )

module.exports = Chat.
```

Step 2: Save the messages in DB

→ write a logic to save the msgs in DB inside a "send Message" event in socket.js file.

eg:

```
socket.on ("sendMessage", async ({firstName, userId, targetUserId,
                                  text}) =>{

    try {
        const roomId = getSecretRoomId(userId, targetUserId)

        //find whether there is a chat present in DB of these Participants
        // if present - push the new chat to it
        // else - Create a new ~~chat~~ empty chat

        let chat = await Chat.findOne({ participants : { $all: [userId,
                          targetUserId] },
                  })

Create     [ if (!Chat){
a new chat [     chat = new Chat ({ participants : [userId, targetUserId],
           [                      messages : [ ]
           [                  })
           [ }

if already exists
Push this new  ← chat.messages.push ({ senderId : userId, text,})
msg to it
        await chat.save() → save it in DB
        io.to (roomId).emit ("messageReceived", {firstName, text})
                     ↳ emitting an event once message
                       Received to indicate frontend.
    } catch(err){
        console.log (err)
    }
})
```

Step3: Create an endpoint to fetch the messages from DB

```
const express = require ("express")

Const Chat = require ("../models/chat")

const chatRouter = express.Router ()

chatRouter .get ("/chat/:targetUserId", userAuth, async (req, res)=>{
    try{
        const userId = req.user._id. → getting userId from userAuth.
        const {targetUserId} = req.params.

        let chat = await chat.findone ({ participants : { $all: [userId, targetUserId]},
                              }).populate ({ path: "messages.senderId",
                                   select: "firstName lastName"
                              })

                    if (! chat) {
 if no     [           chat = new chat ({ participants: [userId, targetUserId],
 chat send [                              messages: []
 an empty  [           ~~await~~. ~~save~~ })
 msg.      [           await chat.save ()
                    }
                    res.json(chat)
    } catch(err) {
        console.log(err)
    }
})
module.exports = chat Router.
```

→ place this router in app.js file.

```
app.use("/", chatRouter).
```

Step 4: Retrieve from DB & display it in UI.

→ create a fn to fetch messages in chat component & update it with state var.

```
const fetchchatMessages = async() =>{
  //make an API call
  const chat = await  axios.get (BackendURL+"/chat/" + targetuserId , {
                          withCredentials: true , })

     const chatmessages = chat?.data?.messages.map((msg)=>{
                   const {senderId, text}= msg
                   return {
                            firstName : senderId?.firstName,
                            text
                          }
     })
     setMessages(chatMessages) → update the state var with the apI response.
}

useEffect(()=>{ fetchChatMessages() }, []) → need to called as soon as Pageload
```

Homework:

1) write a functionality to check whether that 2 person is friend (or) not before sending a msg to avoid manual typing of targetUserId.

2) Display green dot when online & last seen status

3) Limit messages when fetching from DB.

4) Build a tic tac toe (or) chess game (or) type racer.

Step 5: Deploy it to instance:

→ first, update the code in F.E Socket.js file before deploying it to the instance.

```
export const createSocketConnection =()=>{
    if(location.hostname === "localhost"){
        return io(BaseURL)
    }else {
        return io("/", {path: "/api/socket.io"})
    }
}
```

deploy it in instance.

×