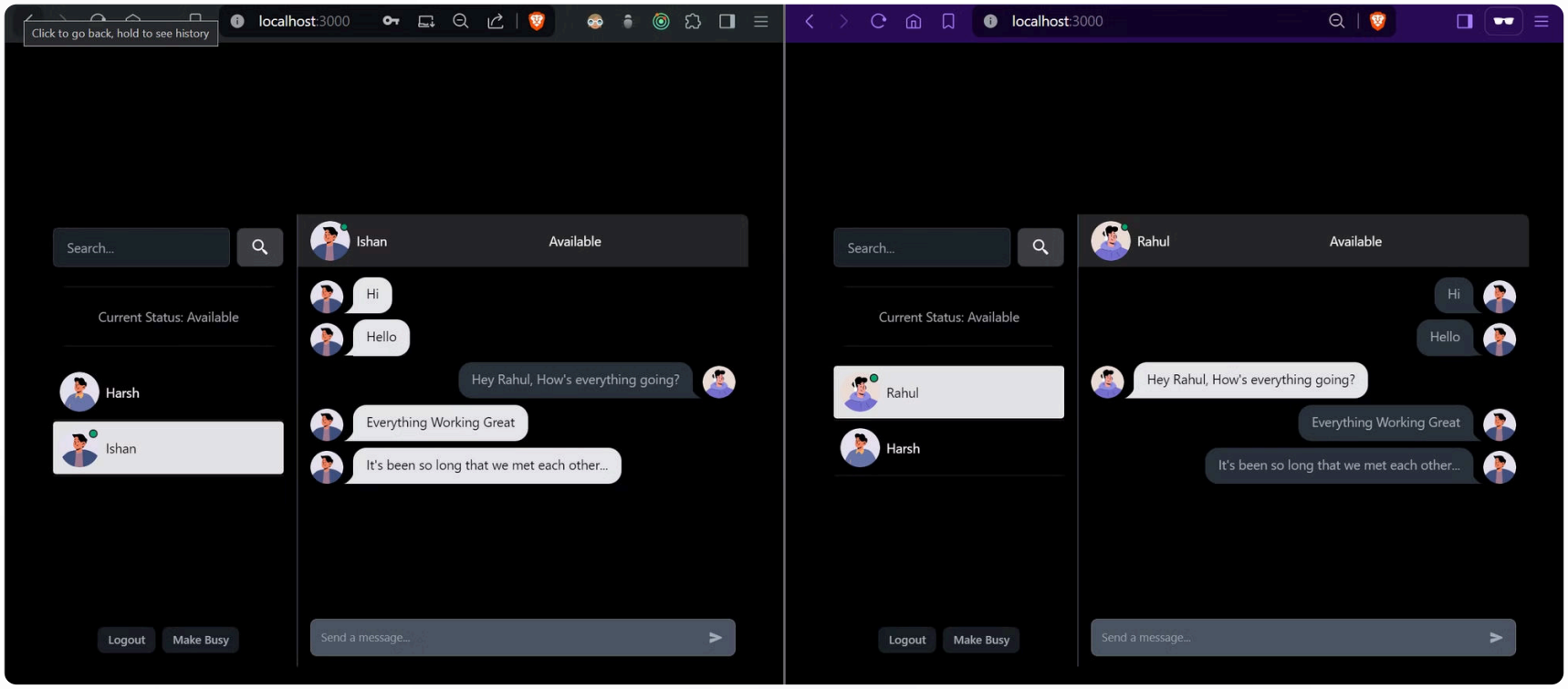
ChatApp_HQ



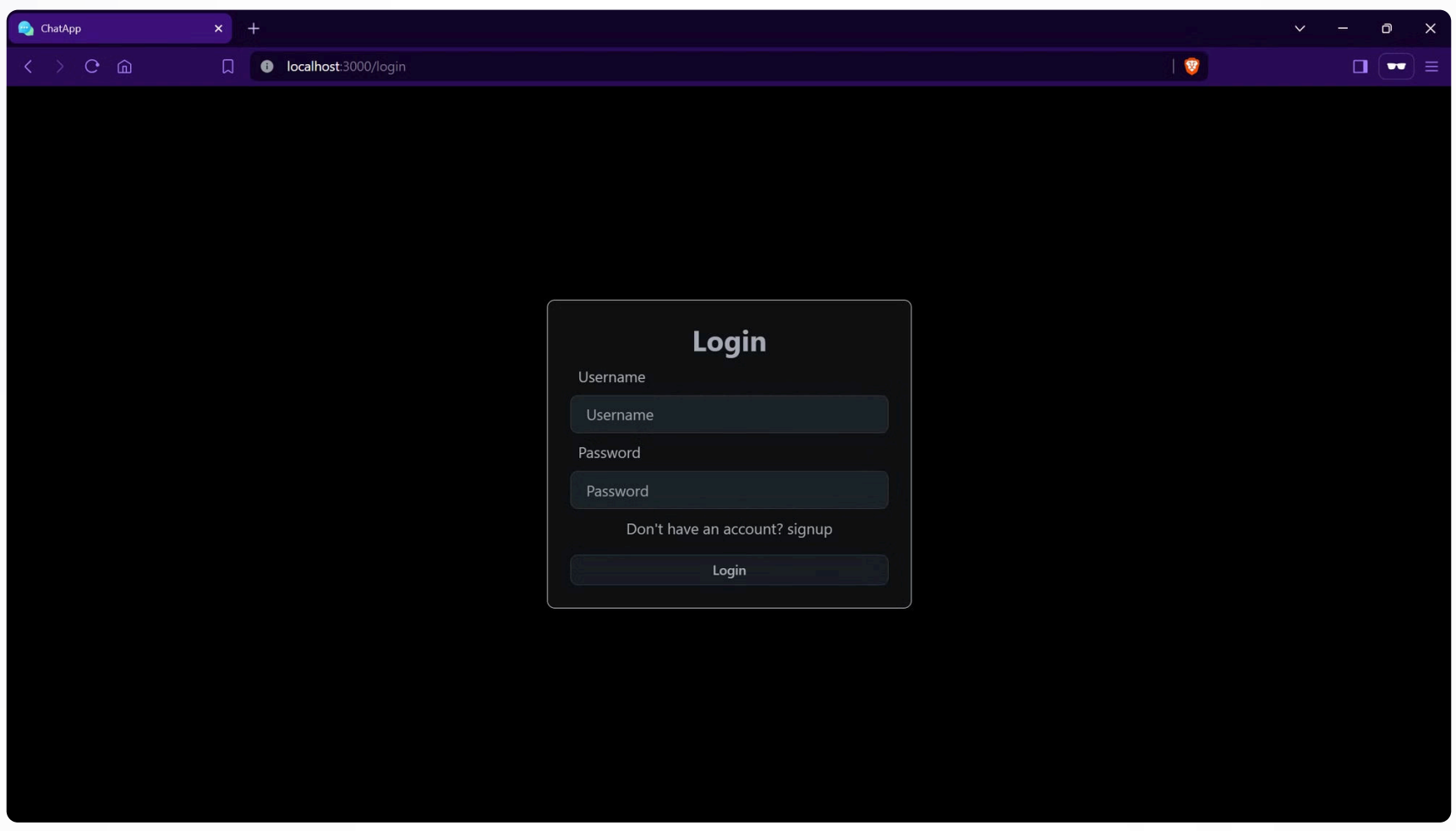
Demo Video:

Google Docs

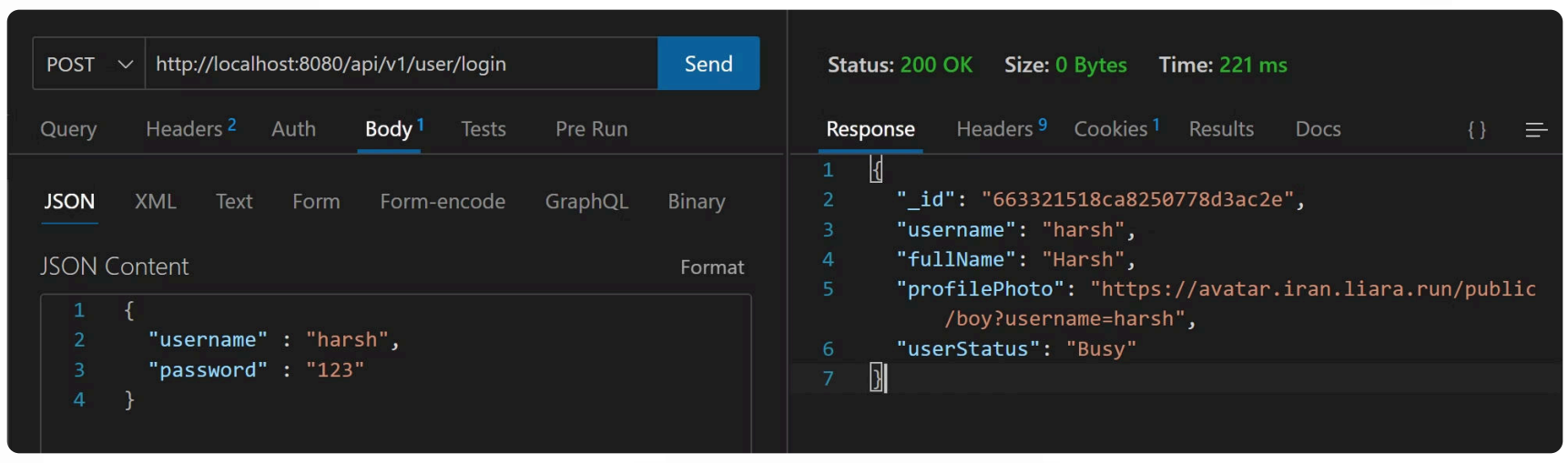
Demo_video_frontend.mp4

This is a video demonstration of my project built for HQ based on Socket.IO, Mern and Jwt Authentication.

When a user lands on a page, first he is referred to login page if he is not logged in, uptil now..

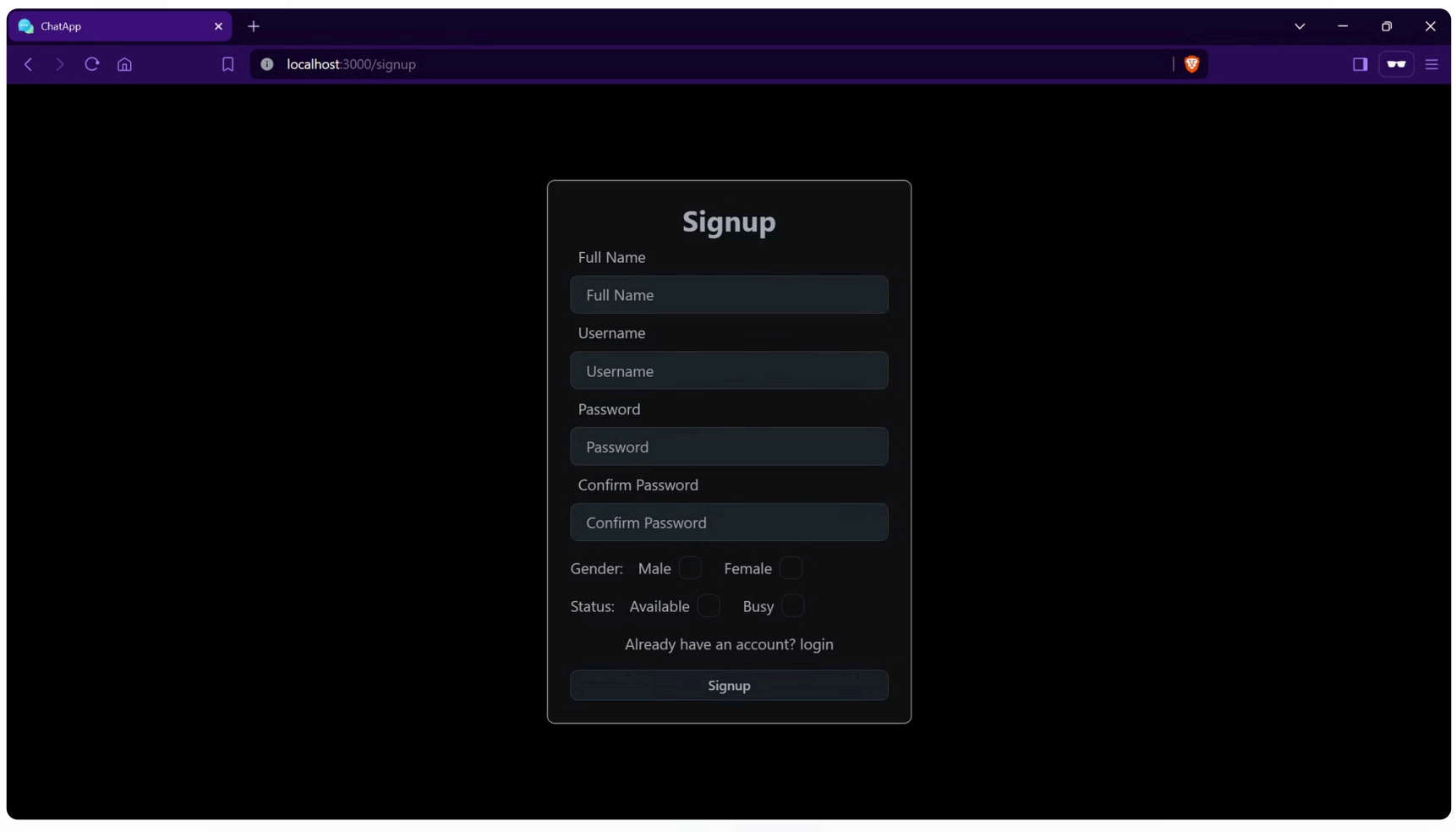


When this Login button is pressed, a **Post** request is being made to
http://localhost:8080/api/v1/user/login
And in request, username and password is passed, for successful user Login...
Picture demonstrating this:



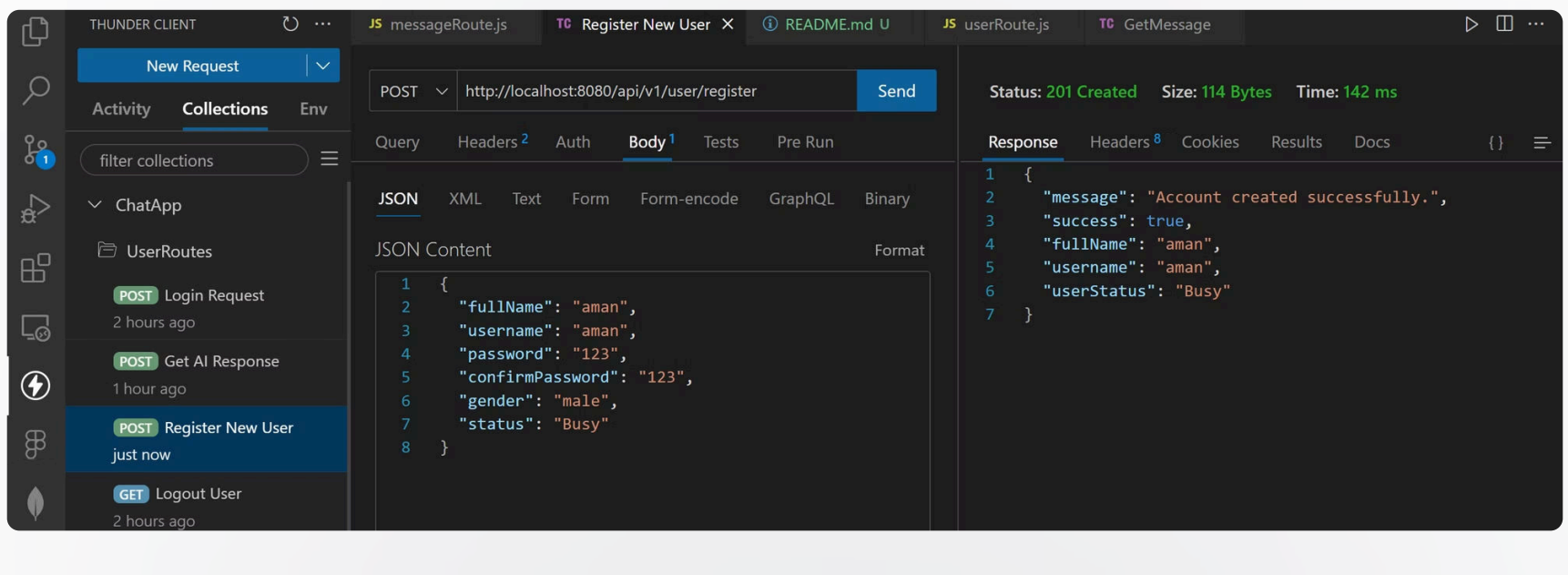
Status Code and Other Details are sent back as a response.

If a User is not a registered User then he/she can register themselves up also.



As it has been done for Login, Similarly it would be the case for Register Page,
a **POST** request to **http://localhost:8080/api/v1/user/register**

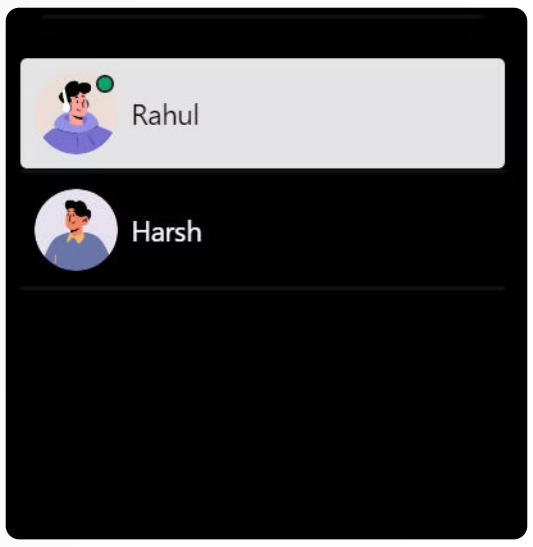
```
//With input in body as:

{
  "fullName": "aman",
  "username": "aman",
  "password": "123",
  "confirmPassword": "123",
  "gender": "male",
  "status": "Busy"
}
```



And that's how authentication is working in this app.

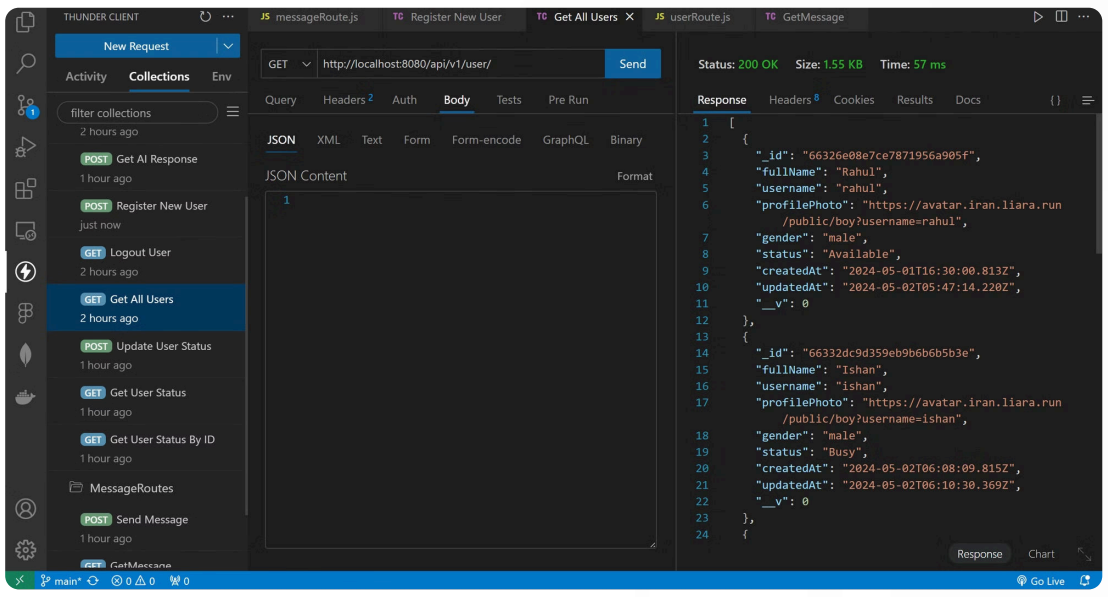
Getting All the users to Chat with..

All users are fetched after first login.



(Did you notice that little green dot there?)

Backend for it:

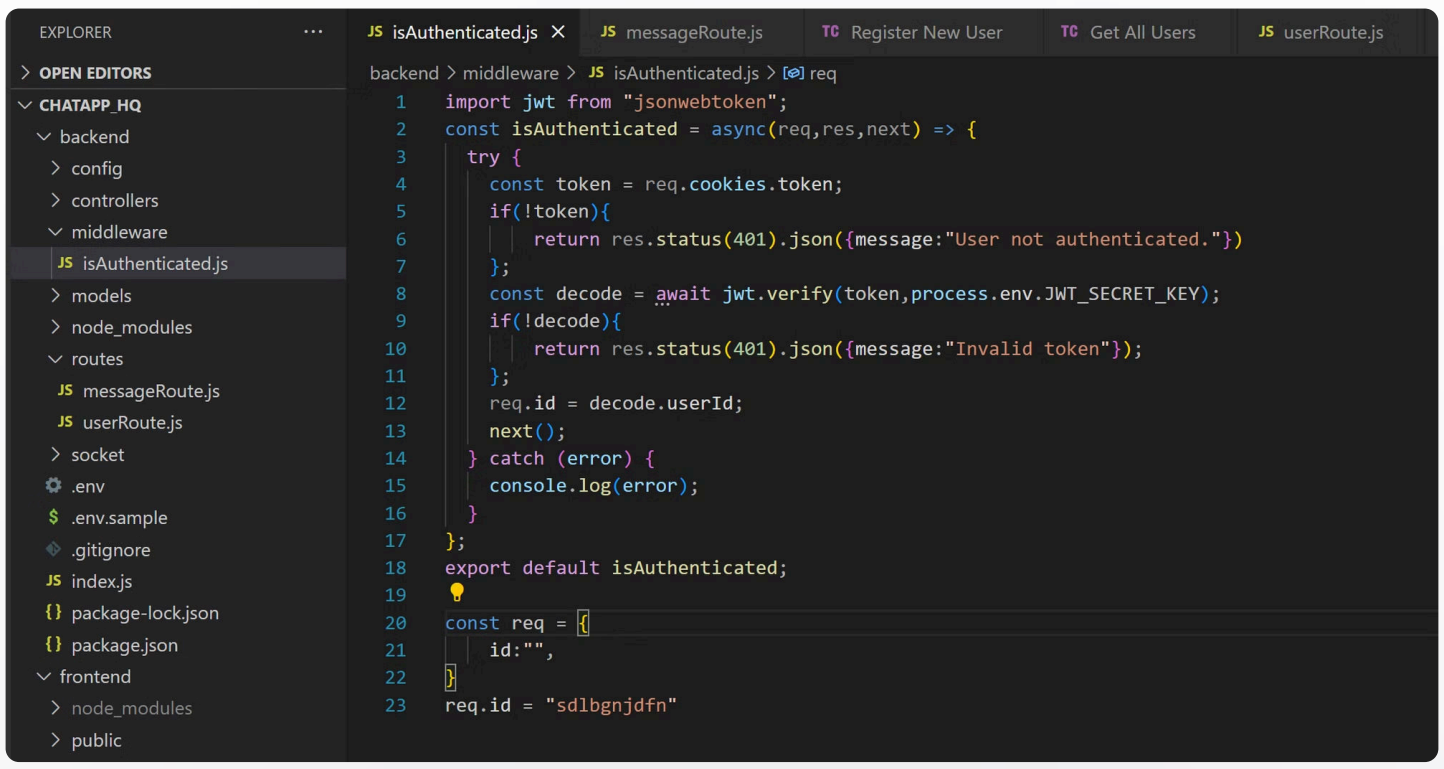


All the users and their details being passed in response, after a get request on /api/v1/user

Uhh!! Let's not forget that anyone can get this detail then because there is no authentication checkup... That's why, while defining UserRoutes, I have passed a middleware check first!

```
import express from "express";
import { generateResponse, getOtherUsers, getStatusByUserId, getUserStatus, login, logout, register,
updateUserStatus } from "../controllers/userController.js";
import isAuthenticated from "../middleware/isAuthenticated.js";

const router = express.Router();

router.route("/register").post(register);
router.route("/login").post(login);
router.route("/logout").get(logout);

router.route("/").get(isAuthenticated,getOtherUsers);

router.route("/updatestatus").post(updateUserStatus);
router.route("/status").get(isAuthenticated, getUserStatus);
router.get('/status/:userId', isAuthenticated, getStatusByUserId);

router.route('/ai/chat').post(generateResponse);

export default router;
```

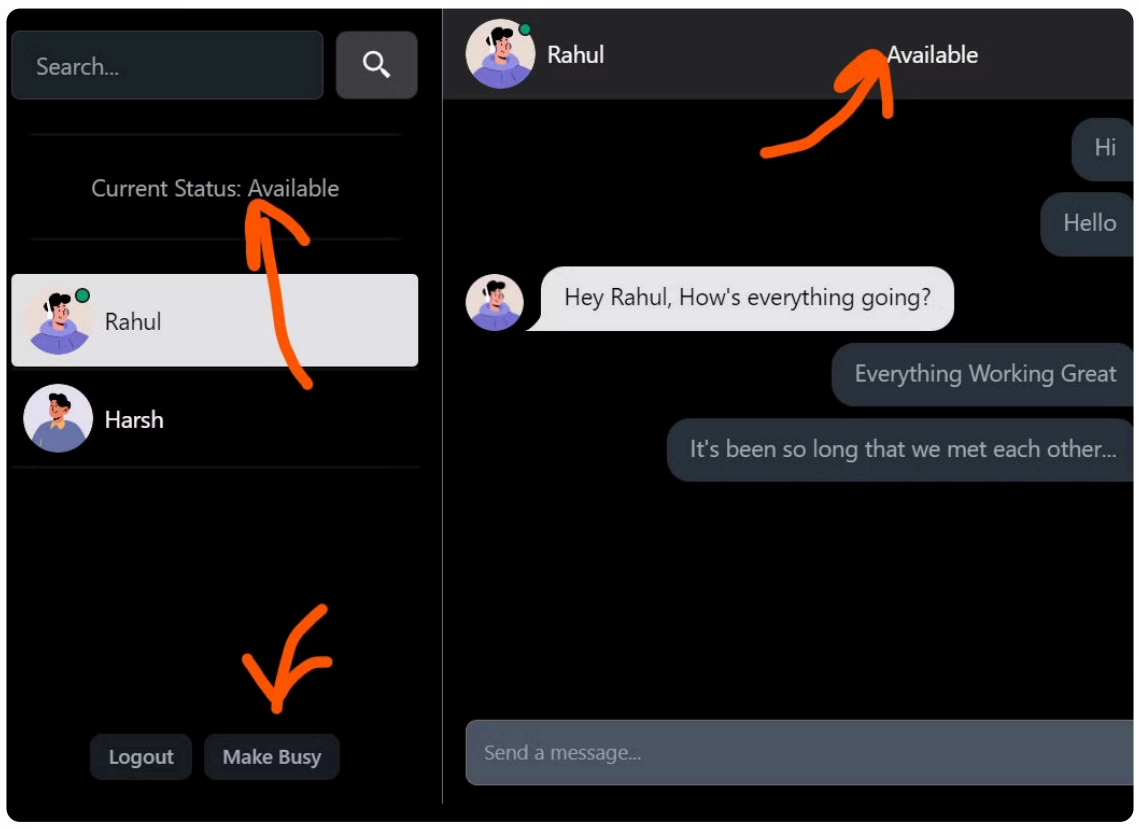
There is this middleware check first code:



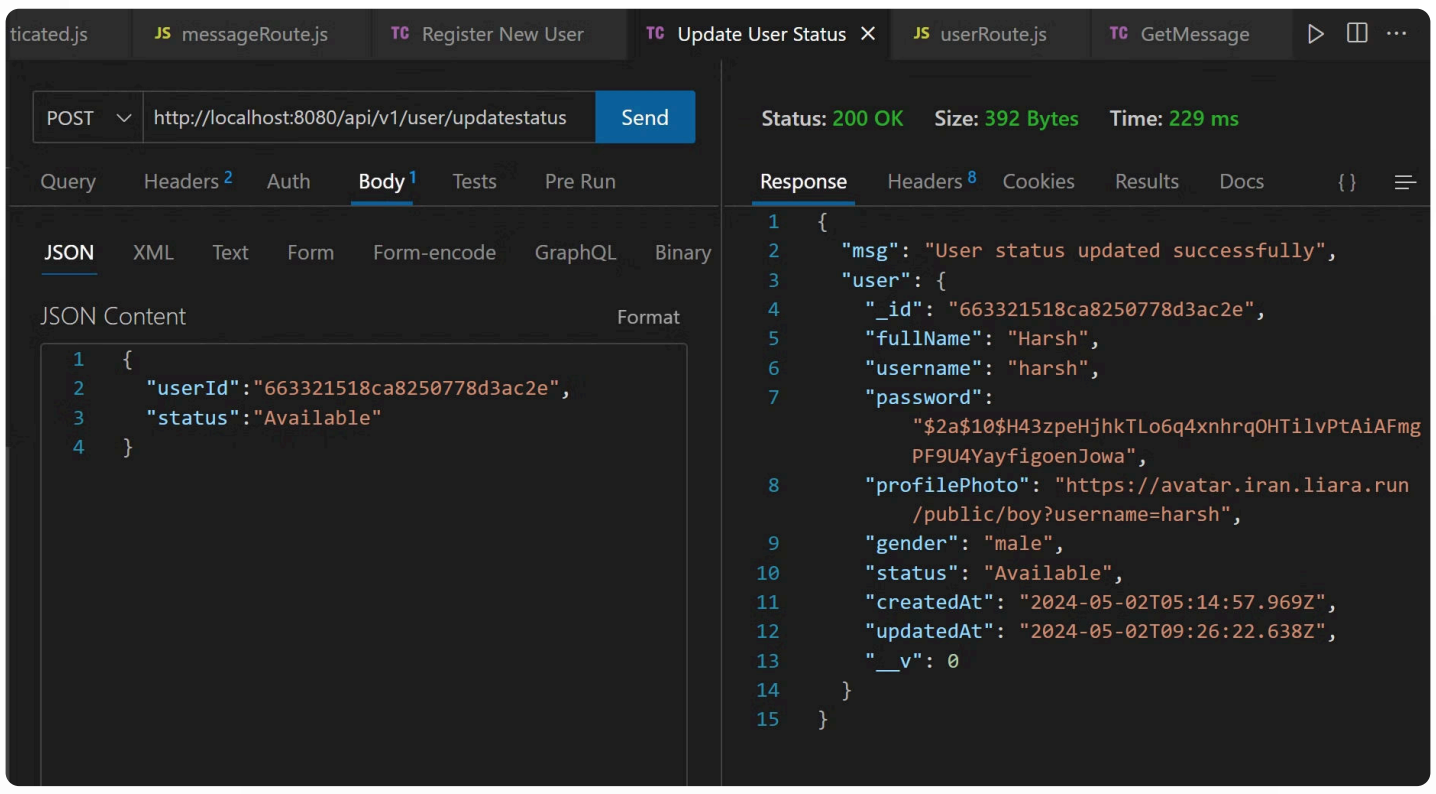
If User is authenticated then only and only he can get all the users, and perform other operations.

Status

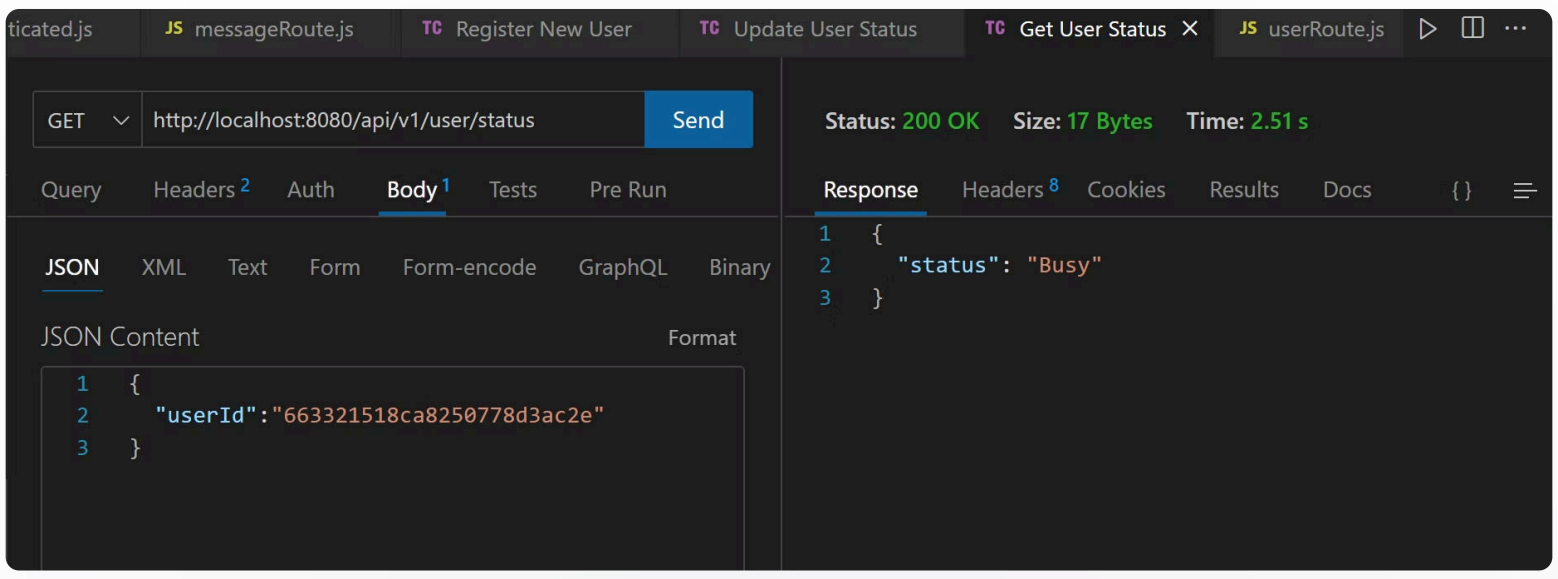
Since it was mentioned that User can Chat if only his status is Available. So to integrate this functionality,
I have created a button, by which a User can alter his status, a Current Status tab at top to show loggedin user's current status. And status of another user is reflected in front of his name...



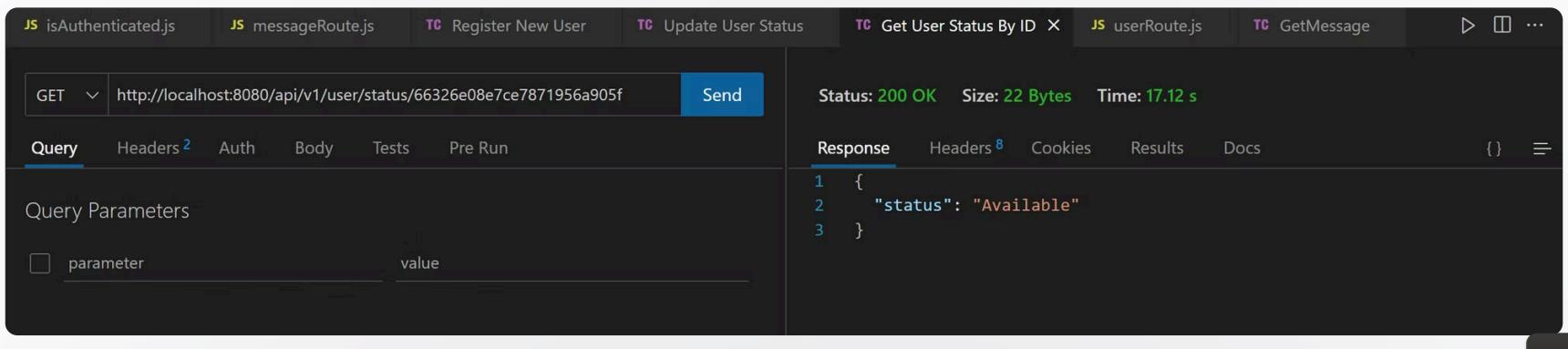
API request to change status of loggedin User:



Similarly, Two more functions/requests are there:
For writing current status of loggedIn User:



And, to get the status of selected User with which we are talking to:
Get User Status By ID, passed in as parameter:



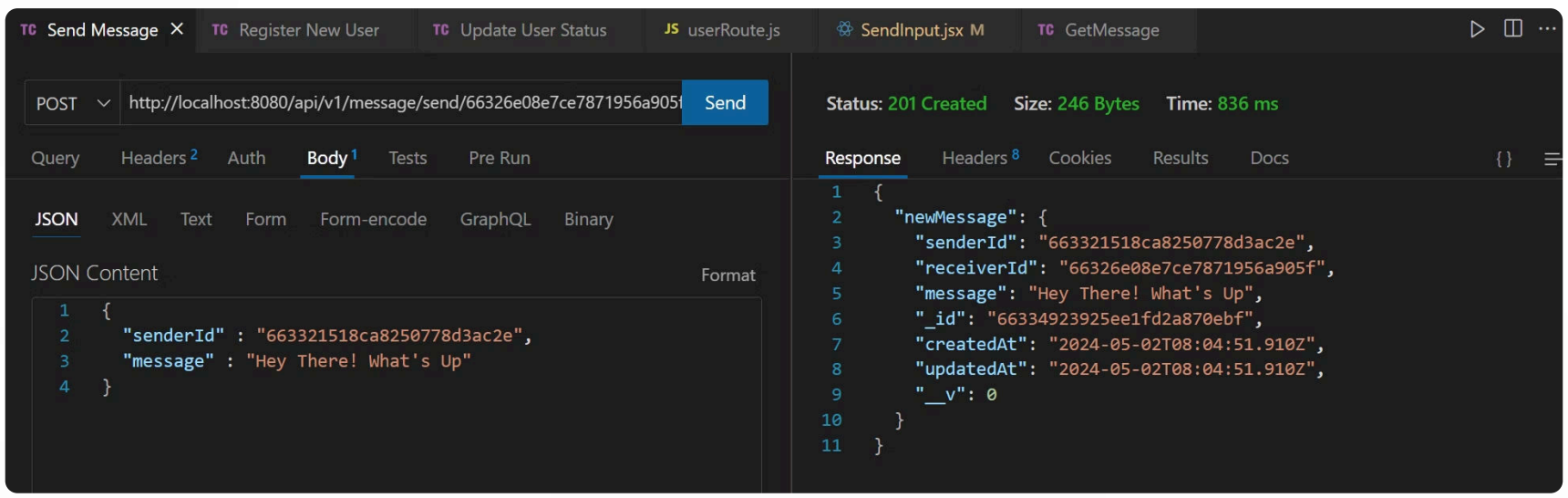
Conversation and Messages API

Message Routes are defined like this:

backend > routes > messageRoute.js > default

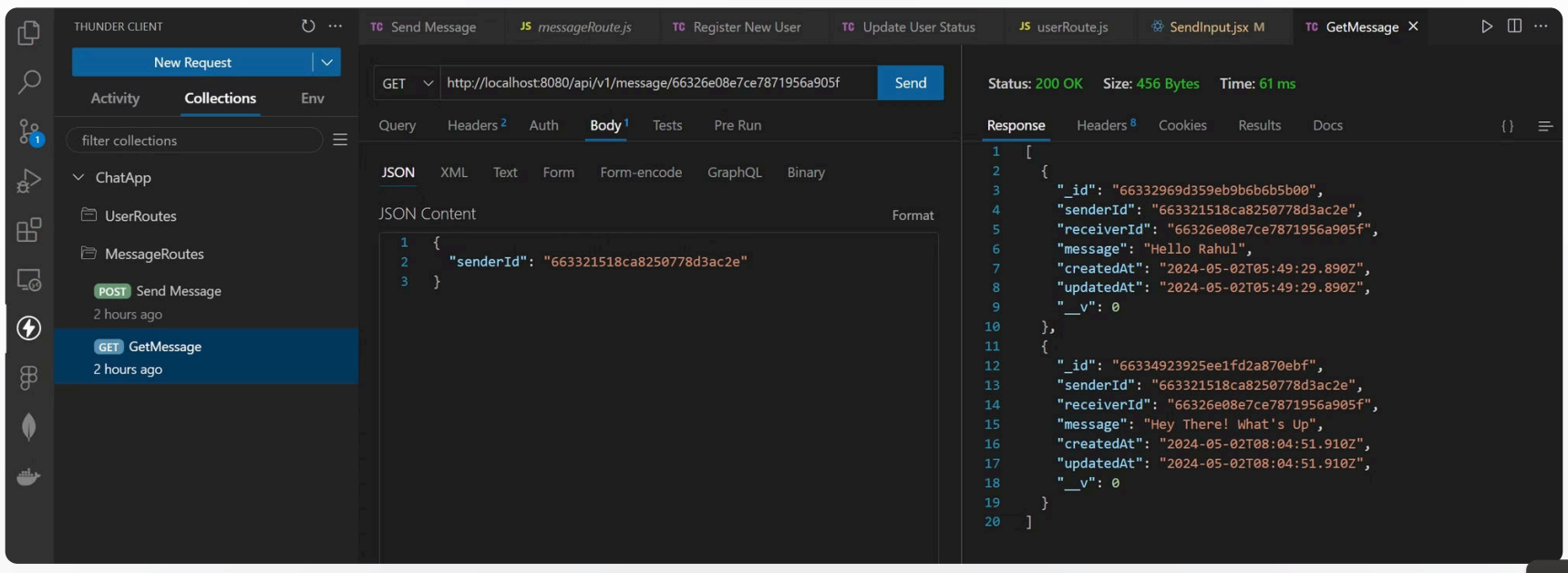
```js
import express from "express";
import { getMessage, sendMessage } from "../controllers/messageController.js";
import isAuthenticated from "../middleware/isAuthenticated.js";

const router = express.Router();

router.route("/send/:id").post(isAuthenticated,sendMessage);
router.route("/:id").get(isAuthenticated, getMessage);

export default router;
```

To send a message from one user to another user,
senderId, recepient Id and message would be required and it would work like this:

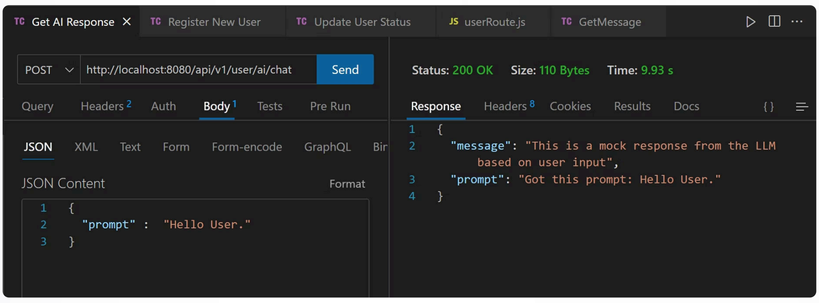


Similarly for **getting all the messages** for a user, a ***GET*** request would be followed on, on this URL:
/api/v1/messages/:id

ThunderClient SS:

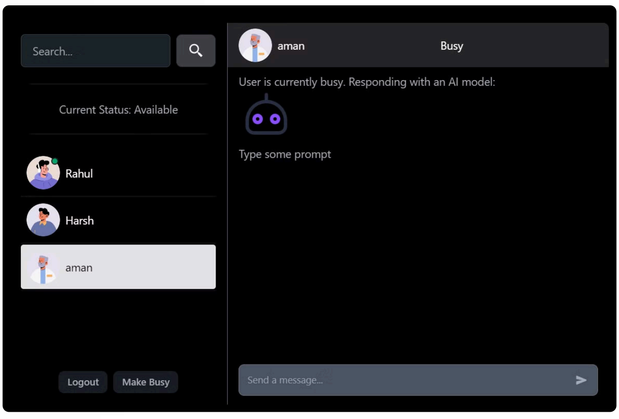


AI Response back:



Whenever a user is busy or not available, then an AI message will respond back to the other user...
How?
let's see

Since Aman has updated his status to "Busy" this means no one can chat to him right now,
That's why,



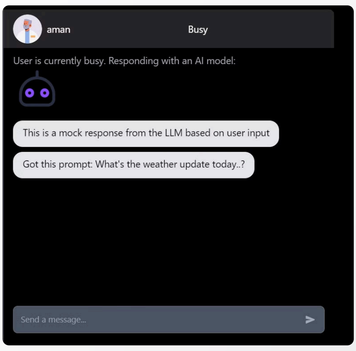
This Ai bot popup would be there instead of messages or chat of the two people...
Since I have used a mockup function instead of real LLM response, response would come in 5 to 15 seconds of interval...

```
export const generateResponse = async (req, res) => {
  try {
    const { prompt } = req.body;
    const timeout = Math.random() * (15000 - 5000) + 5000;
    setTimeout(() => {
      res.status(200).json({
        message: "This is a mock response from the LLM based on user input",
        prompt:`Got this prompt: ${prompt}`
      });
    }, timeout);
  } catch (error) {
    console.error("Error generating response:", error);
    res.status(500).json({ error: "Internal Server Error" });
  }
};
```

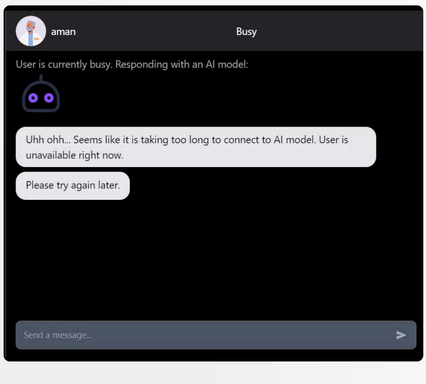
But there is the catch now,
If response takes longer than 10 seconds in total, then: a generic message would be printed instead of actual LLM response: (and this logic has been implemented at client side)

```
if (selectedUser?.status == "Busy") {
  const onSubmitHandler = async (e) => {
    e.preventDefault();
    const timeoutPromise = new Promise((resolve) => {
      setTimeout(() => {
        resolve({ data: { message: "Uhh ohh... Seems like it is taking too long to connect to AI model. User is unavailable right now." } });
      }, 10000);
    });
    try {
      const res = await Promise.race([
        axios.post(
          `${BASE_URL}/api/v1/user/ai/chat`,
          { prompt },
          {headers: {
              "Content-Type": "application/json",},
          }),
        timeoutPromise,
      ]);
      if (res.data && res.data.message === "Uhh ohh... Seems like it is taking too long to connect to AI model. User is unavailable right now.") {
        dispatch(setAiMessages([res.data.message, " Please try again later."]));
      } else {
        dispatch(setAiMessages([res.data.message, res.data.prompt]));
      }
    } catch (error) {
      toast.error(error.response?.data?.error || "An error occurred");
      console.error("Error fetching AI response:", error);
    }
    setPrompt("");
  };
```

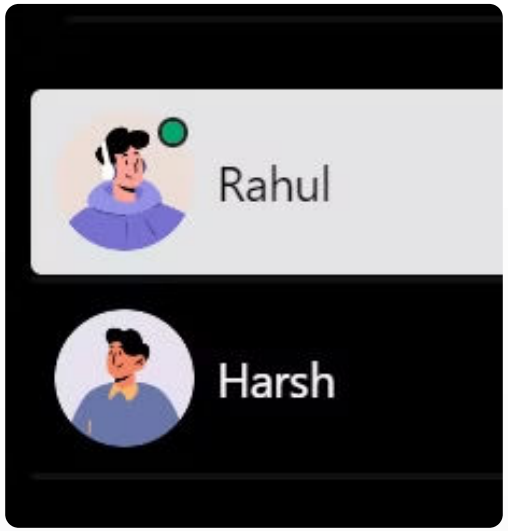
Case 1:
When message takes less than 10 seconds:



Case 2:
When message takes more than 10 seconds:



Also Look at the Green Dot in there, it tells wheather a user is online or not.



So yes that was it, for this project. Hope You like this one.

Github Link: Link
Video Folder Link: Link