Keyboard shortcuts to know

Ctrl + C to quit the running program

What we'll do

In our server folder, we have two key files to add.

The first one is '/server/index.js,' which is responsible for managing the backend of our chatbot. The second file, '/server/api/index.js,' contains the code that handles communication with the API.

When we make requests to the API, whether it's through Postman or our 'App.js' file, those requests are directed to '/server/api/index.js.' This file acts as a bridge, sending the request to the API and then receiving the response. In JavaScript, we use the 'return' statement to send this response back to the original requester, whether it's Postman or our application.

/server index.js

1. Importing the packages/dependencies:

```
import express from 'express';
import http from "http";
import dotenv from 'dotenv';
import api from './api/index.js';
import proxy from "express-http-proxy";
import url from "url";
```

2. Process env file variable for api keys

```
dotenv.config();
```

3. Create express app

```
const app = express();
app.use(express.json());
```

4. Use the imported 'api' for routes under the "/api" path
    - ➢ everything in the api folder connect to the actual app(express app)

```
app.use("/api", api);
```

5. Create a **proxy** for handling requests to a frontend server
    - ➢ A proxy acts like a bridge
    - ➢ It connects your website's frontend (on port 3000) to your backend (on port 3001)
    - ➢ You put your frontend on port 3001 for a specific reason: your server is there, and it helps avoid cross-origin issues (when your domain names are different)
    - ➢ Think of the proxy as a middleman or gateway that ensures smooth communication between your website and the server

```
const feProxy = proxy("http://127.0.0.1:3000", {
    // Define a function to resolve the proxy request path based on the original URL
proxyReqPathResolver: (req) => url.parse(req.originalUrl).path || ""
});
// Use the proxy for all routes (wildcard) to forward requests to the frontend server
app.use("/*", feProxy);
```

6. Define the **port** for express application to listen on

```
const PORT = process.env.PORT || 3001;
```

7. Create an HTTP server

```
// Create an HTTP server using the Express app
const httpServer = http.createServer(app);
// Start the HTTP server, and log a message when it's listening
httpServer.listen(PORT, () => console.log(`Server is listening on Port ${PORT}`));
```

/api index.js

1. Import packages used in this file
    - ➢ In this case, it's express' Router (some part of express idk) and axios

```
import { Router } from "express";
import axios from "axios";
const router = Router();
```

2. Write this code to test your server on Postman (optional)

```
router.get('/', (req, res) => {
    res.send("Hello world!");
})
```

3. This code defines a POST endpoint at `/chat` to handle chatbot requests
    - ➢ When a POST request is made:
        - It extracts the `query` property from the request body
        - Checks if `query` is empty or missing, responding with a 400 Bad Request if so
        - If `query` is valid, it calls `getResponse(query)` to process the user's message
        - It waits for the response from `getResponse` and sends it back to the client (frontend) as the reply to the user's query

```
// Define a POST endpoint for handling chatbot requests
router.post('/chat', async (req, res) => {
    //object destructuring, destructure the 'query' property
from the request body
    const { query } = req.body;

    console.log("got the msg");

    // Check if 'query' is missing or empty
    if (!query) {
        // Send a 400 Bad Request response with an error
message
        res.status(400).send({error:"Query is required"});
    }

        // Call the 'getResponse' function with the 'query'
and send the response
    const response = await getResponse(query);
    res.send(response); //sends response to client i.e.
frontend
})
```

4. Add a getResponse async function, the following code in the function is copied from RoboMatic.AI to make a request with the API

```
// Define an asynchronous function 'getResponse' for making
API requests
const getResponse = async (query) => {
    // Create URL-encoded parameters
    const encodedParams = new URLSearchParams();
    encodedParams.set('in', query);
    encodedParams.set('op', 'in');
    encodedParams.set('cbot', '1');
    encodedParams.set('SessionID', 'RapidAPI1');
    encodedParams.set('cbid', '1');
    encodedParams.set('key', process.env.ROBOMATIC_KEY); //
Set an API key from environment variables
    encodedParams.set('ChatSource', 'RapidAPI');
    encodedParams.set('duration', '1');

    // Define options for the HTTP request
    const options = {
    method: 'POST',
    url: 'https://robomatic-ai.p.rapidapi.com/api',
    headers: {
        'content-type': 'application/x-www-form-urlencoded',
        'X-RapidAPI-Key': process.env.RAPIDAPI_KEY, // Set an
API key from environment variables
        'X-RapidAPI-Host': 'robomatic-ai.p.rapidapi.com'
    },
    data: encodedParams,
    };

    try {
        // Make the HTTP request to the chatbot service API
using axios
        const response = await axios.request(options);
        // Log the response data and return it
        console.log(response.data);
        return response.data;
    } catch (error) {
```

```
        // Handle any errors that occur during the request
        console.error(error);
    }
}
```

5. Export the router as the default export of this file

```
export default router;
```

/server package.json

1. Under “scripts”, add “start”:”nodemon index.js”

```
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

2. Add “type”:”module”, under “license”
   - ➢ The default react code we have downloaded is not recent and adding this is such that we can use ‘import’ to get the packages in our files

   This is how your package.json should look like:

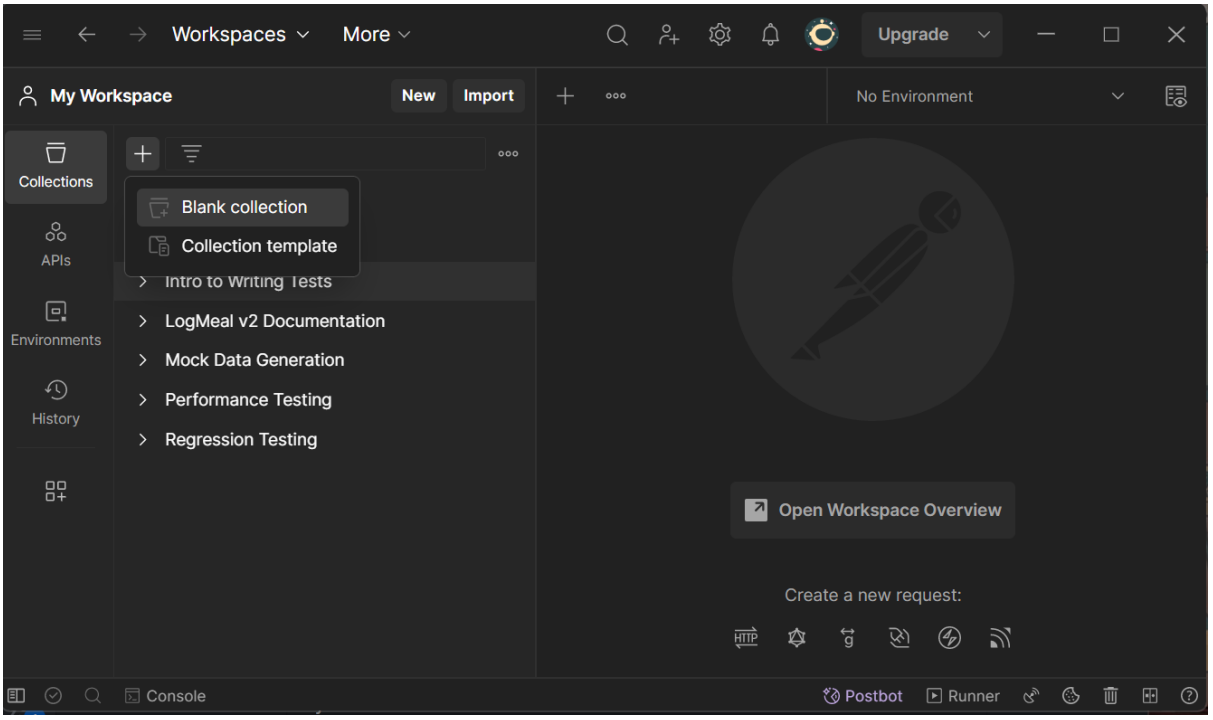
```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "module",
  "dependencies": {
    "axios": "^1.5.1",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "express-http-proxy": "^2.0.0",
    "nodemon": "^3.0.1"
```

```
    }
}
```

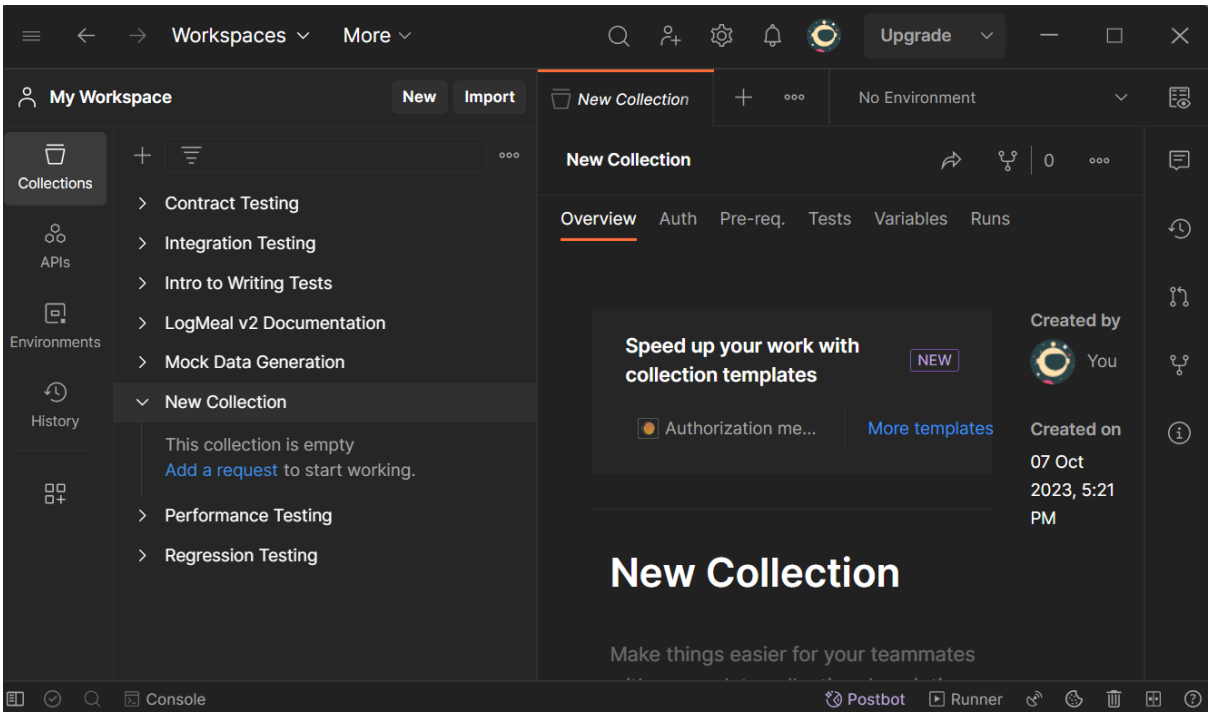
Testing server on Postman

(ensure you have /server/.env file with your API keys)

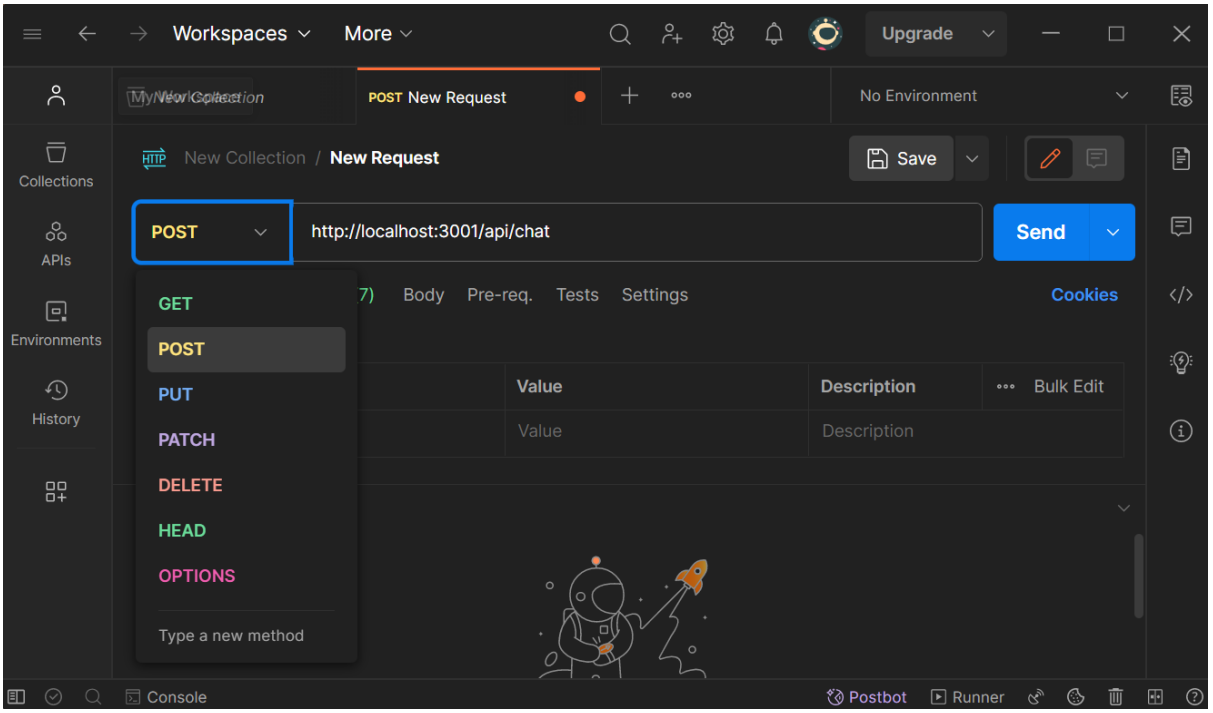
1. On the Postman app, click on Collections on the left side and create a new blank collection:



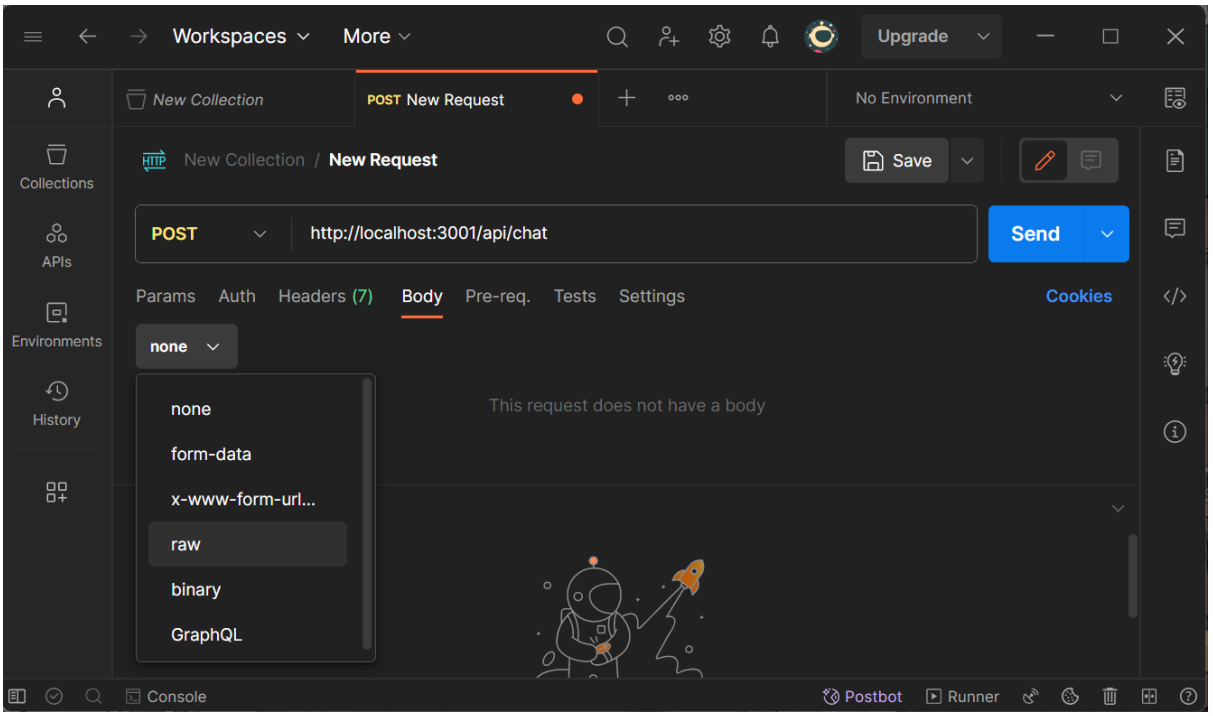
2. Click on Add a request:



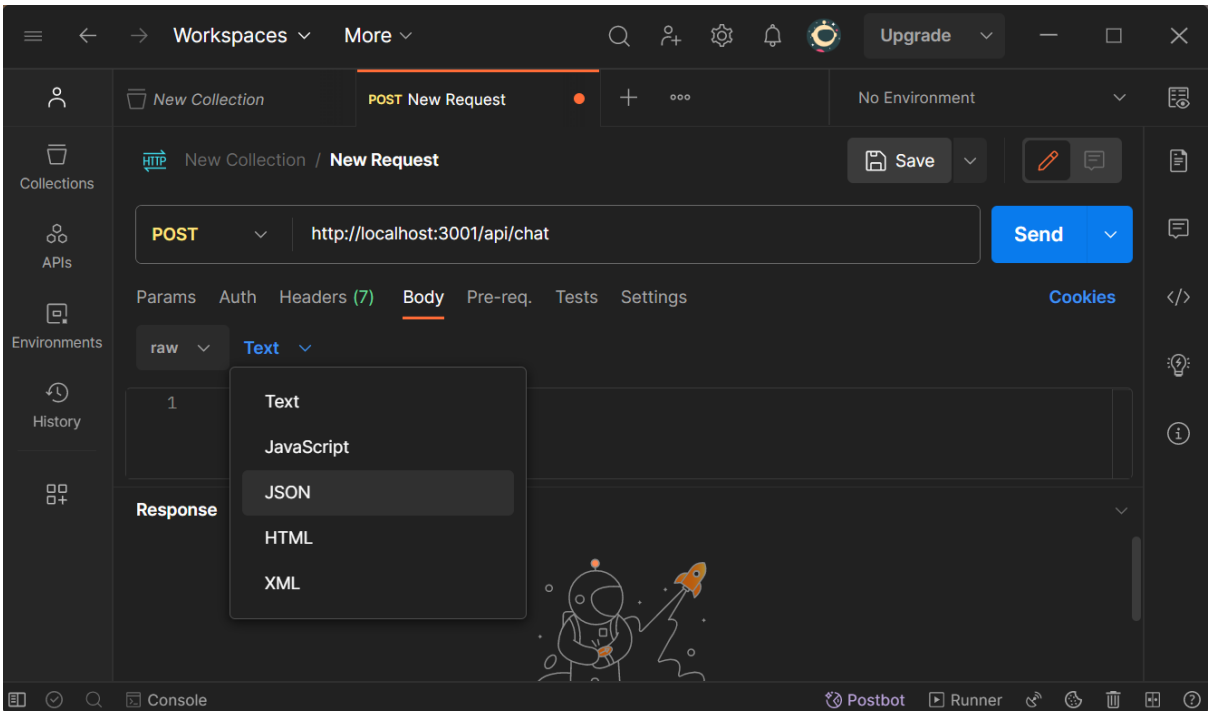
3. Change the GET to a POST request, then add http://localhost:3001/api/chat in the URL:



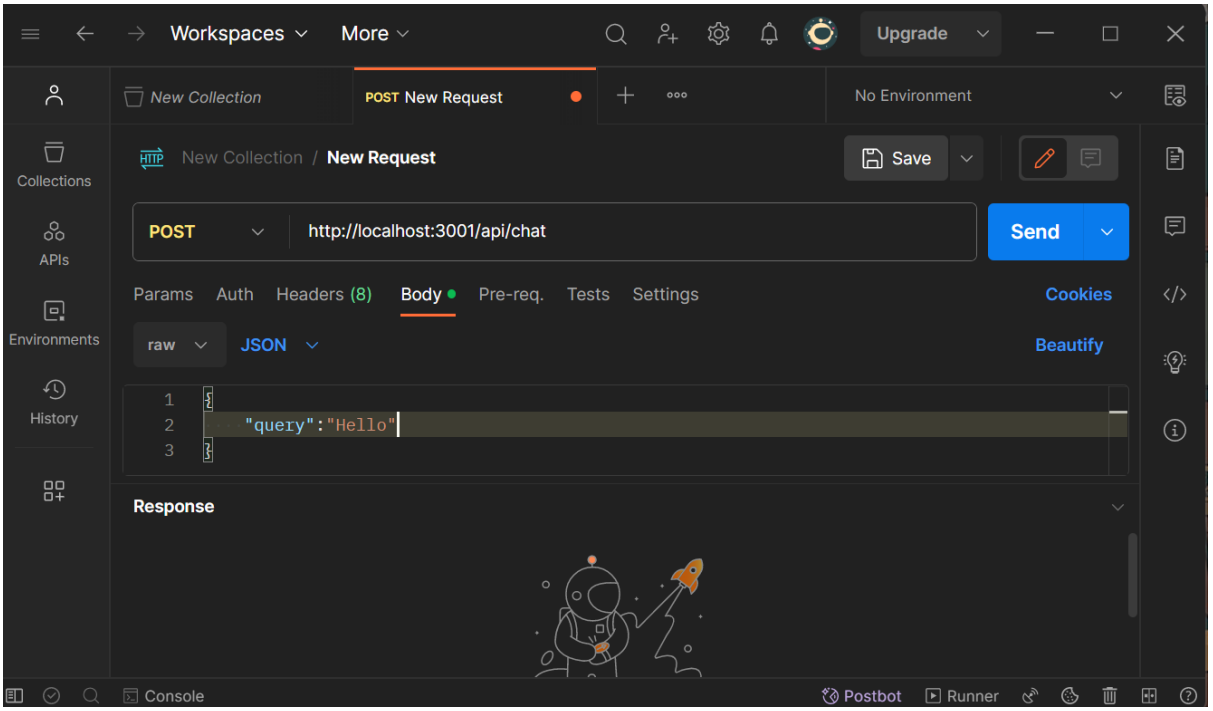
4. Change ‘none’ to ‘raw:

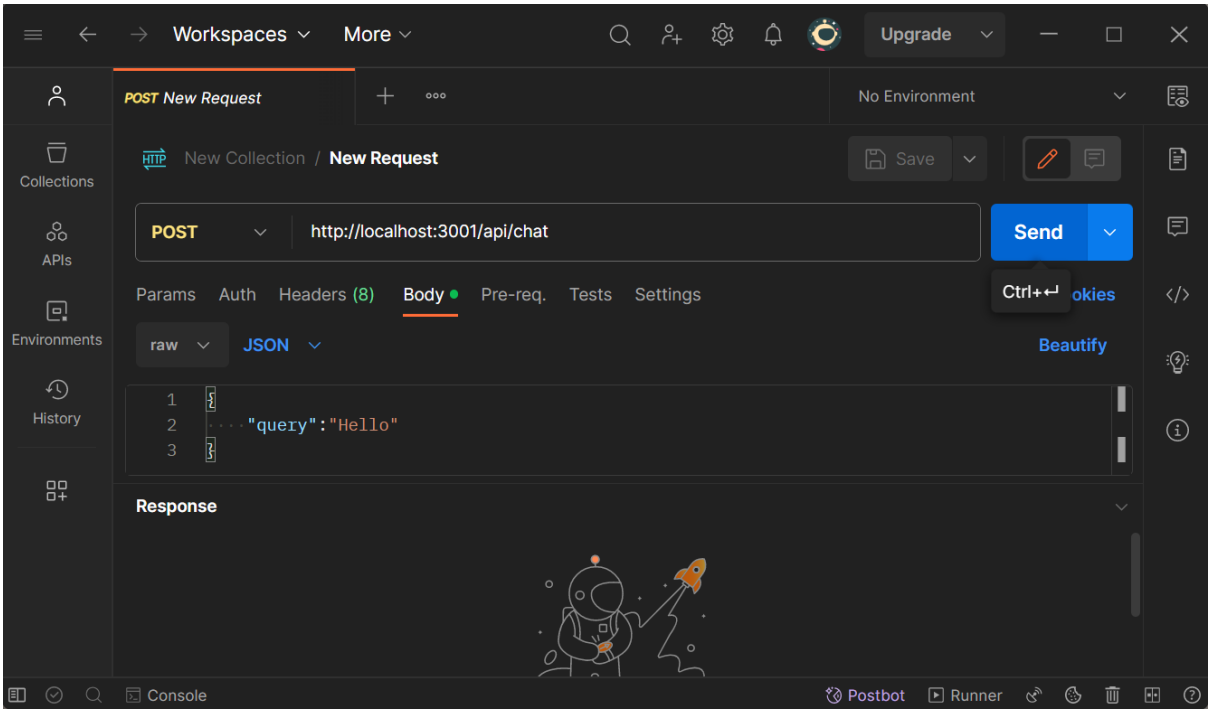


5. And change ‘Text’ to ‘JSON’:

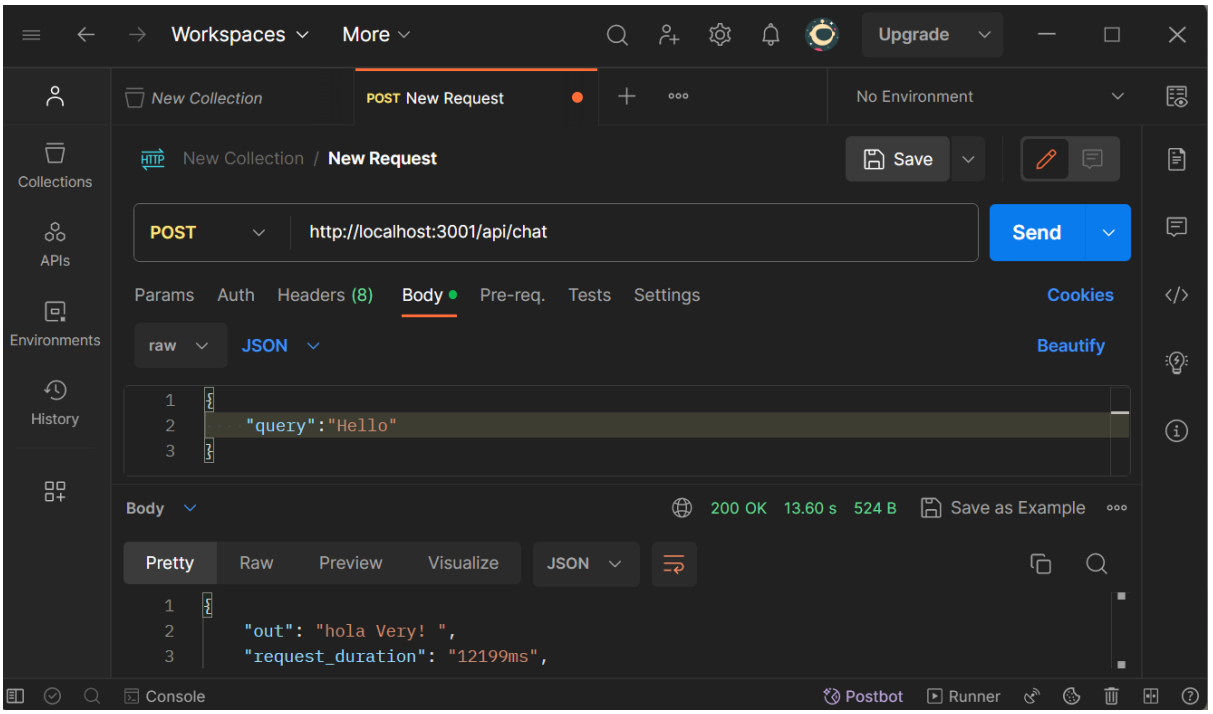


6. This will allow you to enter your query to talk to the bot. Click on Send to send your POST request to the URL:





7. This is what you should see on Postman when the bot API replies:



/client/public index.html

1. (Optional) Change the title of your HTML page

➢ This /client/public/index.html file is your main page that will be loaded. Your chatbot will be rendered to this file using Javascript in /src/App.js file and then /src/index.js file.

/client/src index.js

➢ This section of code renders your App.js as <App/>

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

```
// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more:
https://bit.ly/CRA-vitals
reportWebVitals();
```

/client/src App.js

1. Import relevant packages:

```
import React, { useState } from 'react'; // Import React and the
useState hook from the 'react' library.
import axios from 'axios'; // Import the 'axios' library for making
HTTP requests.
import './App.css'; // Import a CSS file for styling.
```

2. Create a function App(). This function contains the javascript to handle and send a POST request to http://localhost/api/chat and has the HTML code that will be rendered into the /client/public/index.html file.

```
function App() {

  const [message, setMessage] = useState(''); // Create a state
variable 'message' and a function 'setMessage' to update it.
  const [history, setHistory] = useState([]); // Create a state
variable 'history' to store chat history and a function 'setHistory' to
update it.
  const [isLoading, setIsLoading] = useState(false); // Create a state
variable 'isLoading' and a function 'setIsLoading' to track whether a
request is currently loading.

  const handleChat = async () => { // Define an asynchronous function
'handleChat' to handle user interactions.
    try {
      setHistory([...history, { "type": "user", "message": message }]);
// Add the user's message to the chat history.

      setIsLoading(true); // Set 'isLoading' to true to indicate that a
request is in progress.

      console.log("sending");
```

```
      // Make an HTTP POST request to the specified URL with the user's
message as the request payload.
      const response = await
axios.post('http://localhost:3001/api/chat', {
        query: message,
      });

      console.log("send");

      // Extract the bot's response from the HTTP response data, or
provide a default message if there's no response.
      let botMessage = response.data && response.data.out
        ? response.data.out
        : "Sorry, the API is not responding right now";

      // Add the bot's response to the chat history.
      setHistory((prevHistory) => [
        ...prevHistory,
        { "type": "bot", "message": botMessage },
      ]);

      setIsLoading(false); // Set 'isLoading' back to false to indicate
that the request is complete.
      setMessage(''); // Clear the input field by setting 'message' to
an empty string.
    } catch (error) {
      console.error(error); // Log any errors that occur during the
request.
      setIsLoading(false); // Set 'isLoading' to false in case of an
error.
    }
  };

  return (
    <div className="App">
      <h1>Chat Bot</h1> {/* Display a heading for the chat application.
*/}
      <div>
        {/* Display the chat history by mapping through the 'history'
array and rendering each message. */}
        {
          history.map((item, idx) => {
            return (
```

```
              <div className="chat-history" style={{ margin: 10 }}
key={idx}>
                <p>{item.type}:  </p> {/* Display the message type
(user or bot). */}
                <p>{item.message}</p> {/* Display the message text. */}
              </div>
            );
          })
        }
      </div>
      <div className="input-container">
        <input
          type="text"
          value={message}
          onChange={(e) => setMessage(e.target.value)} // Update the
'message' state when the user types in the input field.
        />
        <button onClick={handleChat}>Send</button> {/* Trigger the
'handleChat' function when the "Send" button is clicked. */}
      </div>
      {isLoading ? (
        <div className="loading">Loading...</div> // Display a loading
message when 'isLoading' is true.
      ) : (
        <></> // Render an empty fragment when 'isLoading' is false.
      )}
    </div>
  );
}
```

3. Export the 'App' component as the default export:

```
export default App;
```

/client/src App.css (optional for styling)

```
.App {
  text-align: center;
}
.chat-history {
  display: flex;
  flex-direction: row;
```

```
  justify-content: center;
}
```

Running the chatbot and testing frontend

- Run ‘npm start’ in both folders’ terminal
- In http://localhost:3001/ you will see the chatbot frontend and communicate there. Note that when you do ‘npm start’ in the client folder you will be sent to http://localhost:3000/ and due to domain issues you cannot communicate with the API there.

GitHub - .gitignore file (optional)

If you wish to publish your bot to GitHub, remember to add a .gitignore file outside of the client and server folder, and enter ‘.env’ inside it.