**Server Setup**

**📁 Step 1: Initialize Node Project in Parent Folder**

**Command:**

npm init -y

**Explanation:**

* This command creates a package.json file with default settings.
* It sets up the parent folder as a Node.js project, which is necessary before installing any dependencies.

📌 **Make sure you're in the root (parent) folder before running this command.**

**📁 Step 2: Install Express**

**Command:**

npm install express

**Explanation:**

* Installs **Express**, a minimal and flexible Node.js web application framework.
* It helps to create a backend/server for your React app.
* Adds Express as a dependency in package.json.

✅ Now you're ready to create your server file.

**⚙️ Step 3: Install Dev Dependencies: concurrently and nodemon**

**Command:**

npm install -D concurrently nodemon

**Explanation:**

* -D or --save-dev means these packages are installed as **dev dependencies** (used during development only).
* **concurrently**: Allows you to run multiple commands (like client and server) **at the same time**.
* **nodemon**: Automatically restarts the server when file changes are detected.

🚀 This helps streamline development with automatic reloads and parallel processes.

**📝 Step 4: Modify package.json Settings**

Open your package.json and update the following fields:

{

"description": "Backend server for React app",

"main": "server.js",

"scripts": {

"server": "nodemon server.js"

},

"author": "Joyston"

}

**Explanation:**

* "description": Short info about your project (optional but useful).
* "main": "server.js": Entry point of your server.
* "scripts.server": Defines a command to run the server using nodemon.
* "author": "Joyston": (Optional) Credits you as the author. Not required for functionality, but good for documentation.

✅ Save the file after making these changes.

**🛠️ Step 5: Create server.js and Set Up Express Server**

📍 **Location:** In the **parent folder**, create a file named server.js.

🧾 **Code:**

const express = require("express");

const app = express();

const PORT = process.env.PORT || 8000;

// Routes

app.get("/", (req, res) => {

res.send("Welcome to the BlogVerse Express Server!");

});

// Start server

app.listen(PORT, () => {

console.log(`🚀 Server is running on http://localhost:${PORT}`);

});

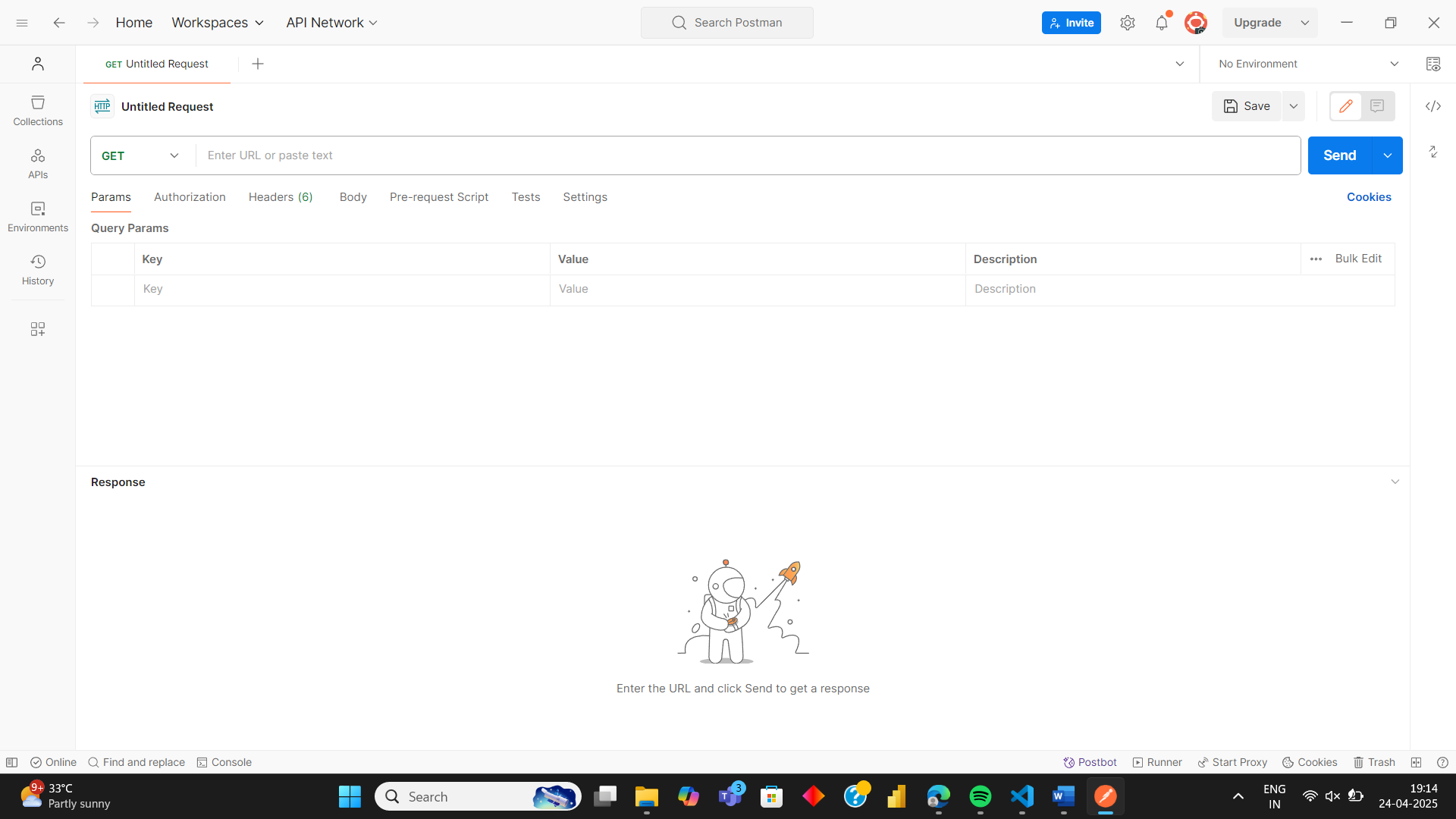
**Explanation:**

* Imports Express and initializes the app.
* Sets a port (from environment or defaults to 8000).
* Defines a simple route for / that returns a welcome message.
* Starts the server and logs a message when it's running.

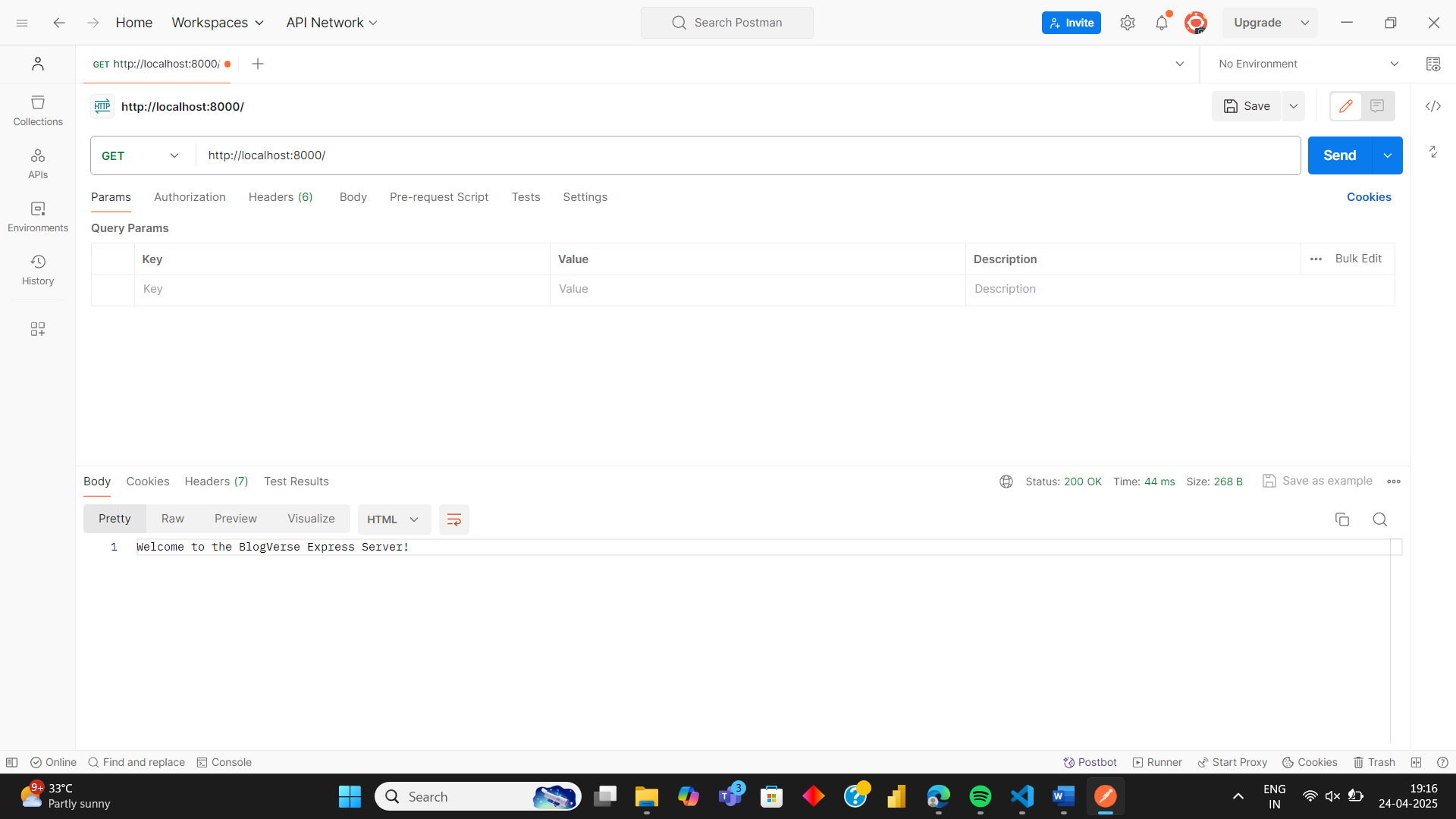
✅ You can now run this server using:

npm run server

**🧪 Step 6: Test the Server with Postman**

1. **Open Postman** on your computer.
2. **Click on the "API" tab** (if it’s not already selected).
3. **Click on the + icon** to create a new request.
4. Set the request type to **GET** (from the dropdown next to the URL bar).
5. **Enter the URL**:  
   http://localhost:8000/
6. **Click the "Send" button**.

**Expected Outcome**

You should see the message **"Welcome to the BlogVerse Express Server!"** in the response body.

**🧪 Step 7: Test POST Request with Postman and Modify server.js**

1. **Modify server.js**:
   * Add middleware to parse JSON request bodies.
   * Update your POST route to handle requests with a name.

**RED-OLD CODE , BLUE-NEW CODE ADDED**

const express = require("express");

const app = express();

const PORT = process.env.PORT || 8000;

// Middleware to parse JSON

app.use(express.json({ extended: false }));

// Routes

app.get("/", (req, res) => {

res.send("Welcome to the BlogVerse Express Server!");

});

// POST route

app.post("/", (req, res) => {

res.send(`Welcome ${req.body.name} to the BlogVerse Express Server!`);

});

// Start server

app.listen(PORT, () => {

console.log(`🚀 Server is running on http://localhost:${PORT}`);

});

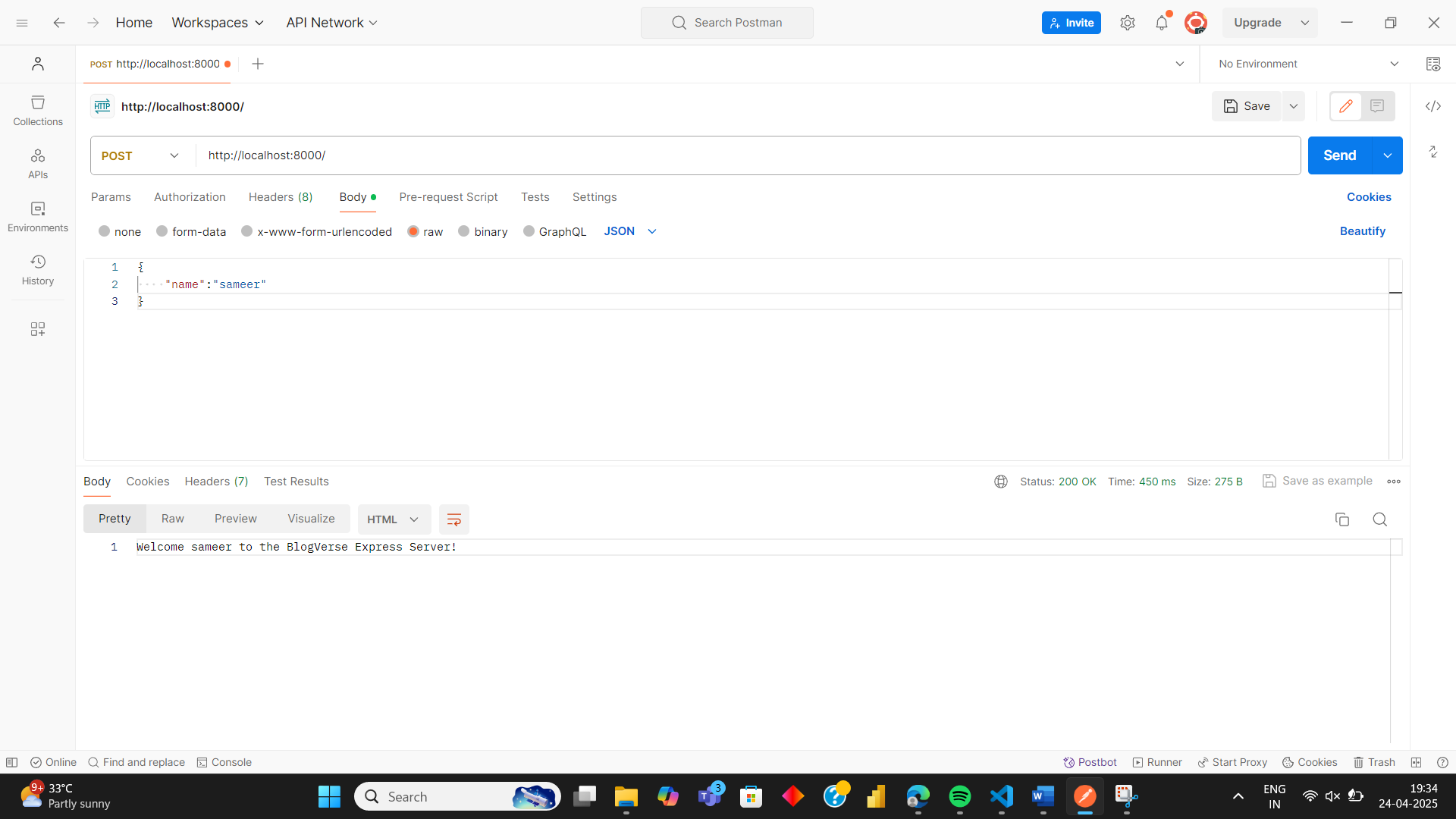
**Explanation:**

* app.use(express.json({ extended: false })): This middleware is added to parse JSON data in the request body.
* The app.post() method now receives data (e.g., {"name": "Sameer"}) and sends a personalized response.

1. **In Postman**:
   * **Select** the **POST** method.
   * **Click on the "Body" tab**.
   * Choose **raw** and then select **JSON** from the dropdown next to it.
   * **Enter this JSON data**:
2. {"name": "Sameer"}
   * **Click the "Send" button**.

**Expected Outcome:**

* The server should respond with:  
  Welcome Sameer to the BlogVerse Express Server!



**🧭 Step 8: Using Route Parameters in Express**

1. **Update server.js** with a new route that uses route parameters:
2. **Add below code ,don’t make changes to other code.**

app.get("/hello/:name", (req, res) => {

res.send(`Hello ${req.params.name}, welcome to the BlogVerse Express Server!`);

});

**Explanation:**

* :name is a **route parameter** — it captures whatever value is passed in the URL.
* req.params.name will retrieve that value.

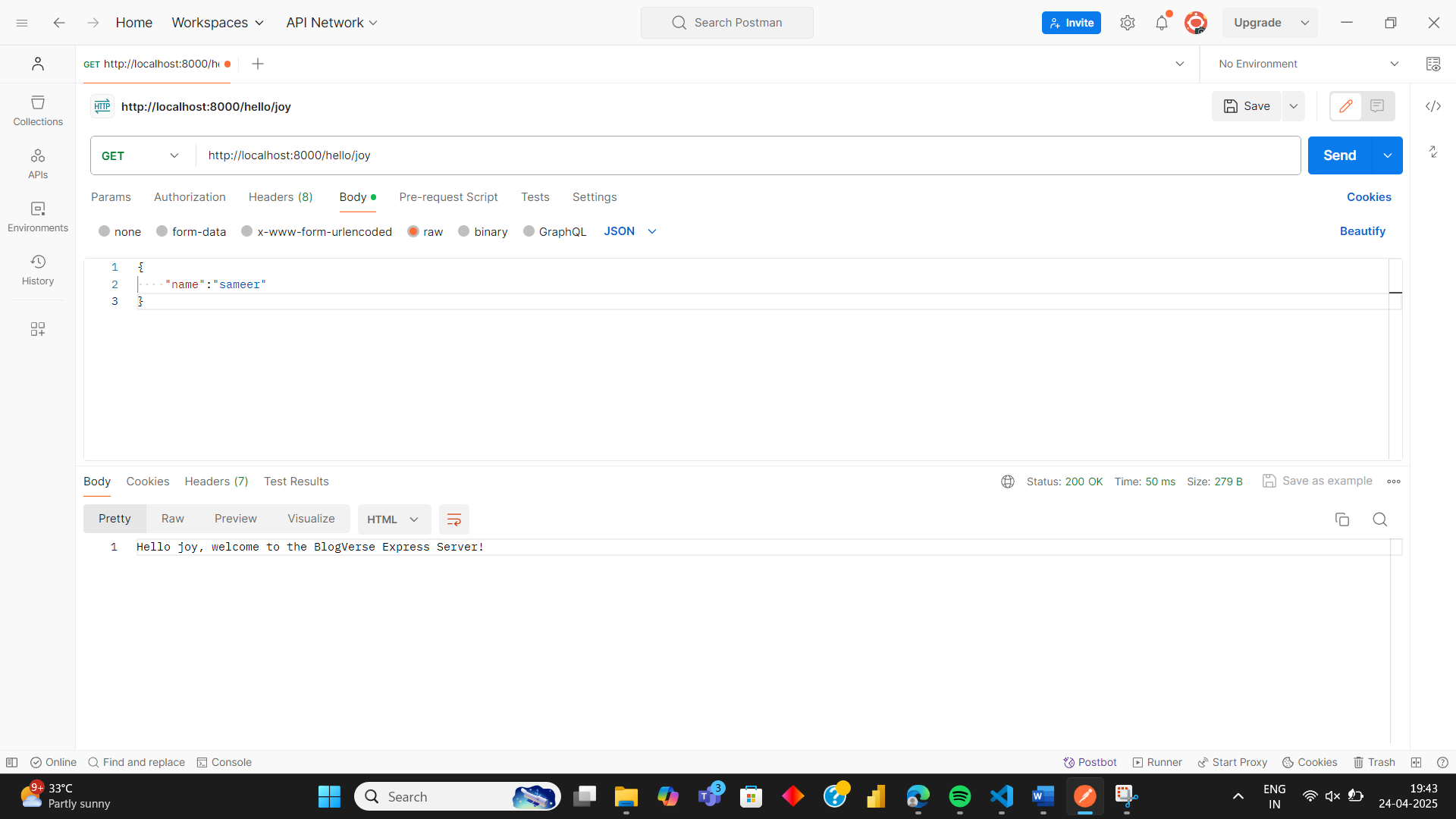
1. **In Postman**:
   * Select the **GET** method.
   * Enter the URL:
   * http://localhost:8000/hello/joy

(You can replace joy with any name.)

* + Click **Send**.

**Expected Outcome:**

* Response will be:  
  Hello joy, welcome to the BlogVerse Express Server!



**💬 Step 9: Add Comments Functionality with Route Parameters**

1. **Update server.js** — Add the following code to simulate a comment system for different articles:

let articleInfo = {

'react-basics': {

title: 'React Basics',

comments: []

},

'node-guide': {

title: 'Node.js Guide',

comments: []

},

'express': {

title: 'Express Tutorial',

comments: []

}

};

app.post('/api/articles/:name/add-comments', (req, res) => {

const articlename = req.params.name;

const { username, text } = req.body;

const article = articleInfo[articlename];

// Push new comment to the appropriate article

article.comments.push({ username, text });

// Return updated article data

res.status(200).send(articleInfo[articlename]);

});

**Explanation:**

* A sample in-memory database articleInfo holds article details.
* Route: POST /api/articles/:name/add-comments accepts article name as a **route parameter**.
* It then adds a new comment (username + text) to the specified article.
* The updated article (with new comment) is returned in the response.

1. **In Postman**:
   * Choose the **POST** method.
   * URL:
   * http://localhost:8000/api/articles/react-basics/add-comments
   * Click on the **Body** tab, select **raw**, and choose **JSON** from the dropdown.
   * Paste this JSON:

{

"username": "sameer",

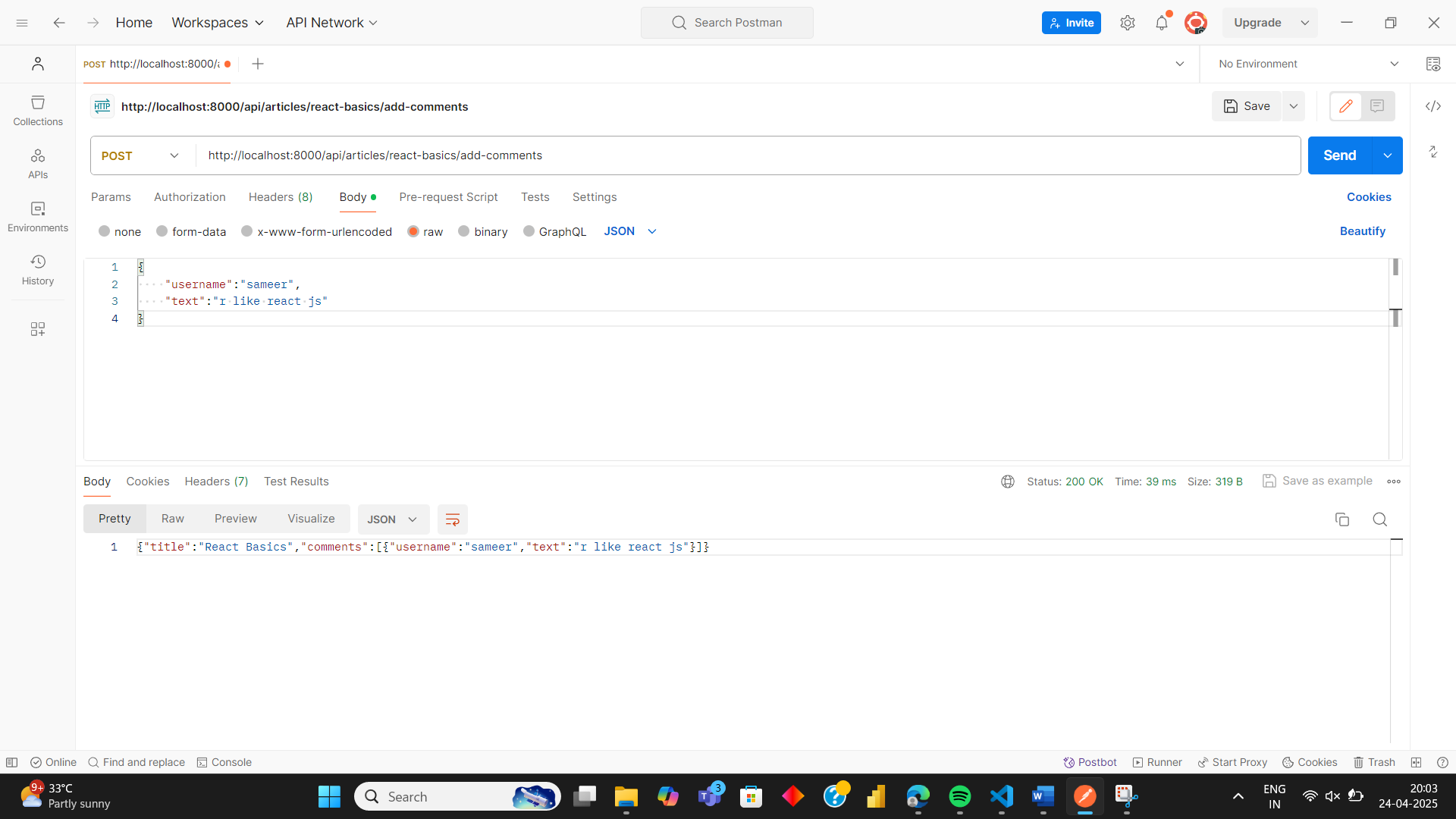
"text": "r like react js"

}

* + Click **Send**.

**Expected Outcome:**

* You’ll receive the updated react-basics article object with the new comment inside the comments array.



**MONGODB**

**Install MongoDB Locally**

**✅ Install MongoDB on your system:**

* [Download MongoDB Community Server](https://www.mongodb.com/try/download/community)
* Install it and make sure mongod is running.

**✅ Create a database:**

1. Open terminal and run:

mongosh

1. Create/use your DB:

use blogverseDB

1. Insert a test document:

db.articles.insertOne({ title: "Hello MongoDB", comments: [] })

Perfect! Let’s document this step in your notes. Here’s the next section:

**💾 Step 10: Install MongoDB Driver in Your Project**

Now that you're ready to connect MongoDB with your Express server, you need to install the official **MongoDB Node.js driver**.

**🛠️ In your terminal (inside your main project folder), run:**

npm install --save mongodb

**✅ What this does:**

* Downloads the MongoDB driver library.
* Lets you connect to a MongoDB database from your Express app.
* Allows operations like inserting, updating, finding, and deleting data in MongoDB.

Would you like to continue with:

1. Creating a MongoDB database & collection,
2. Connecting to it in server.js,
3. Or using Mongoose (a simpler way to work with MongoDB in Node.js)?

Let me know — we’ll take the next step together!

Here’s how this step fits into your notes and what it’s doing:

**🧩 Step 11: Fetch Article Data from MongoDB**

**📄 In server.js, add this code at the top (with your other routes):**

const { MongoClient } = require("mongodb");

app.get("/api/articles/:name", async (req, res) => {

try {

const articlename = req.params.name;

const client = await MongoClient.connect("mongodb://localhost:27017");

const db = client.db("blogverse");

const articleInfo = await db

.collection("articles")

.findOne({ name: articlename });

res.status(200).send(articleInfo);

client.close();

} catch (error) {

console.error("Error fetching article:", error);

res.status(500).send("Internal Server Error");

}

});

**⚠️ Fix small typo:**

Make sure it’s await db.collection(...), **not** awaitdb.collection(...).

**🧪 Step 12: Test the API in Postman**

1. Open **Postman**.
2. Choose GET method.
3. Enter URL:
4. http://localhost:8000/api/articles/react-basics

*(Replace "react-basics" with the actual name in your DB if needed.)*

1. Click **Send**.

✅ If everything is set up correctly:

* You’ll get a JSON response with the article data from MongoDB!

**🚀 What happened here?**

* MongoClient.connect(...) connects to MongoDB.
* db.collection("articles").findOne(...) fetches a specific document.
* The response is sent back to the browser or Postman.

Let me know when you're ready to:

* Insert new articles into MongoDB,
* Add Mongoose to simplify database work, or
* Create routes for updating/deleting articles.

You're building a solid **full-stack backend** — nice work!

**Connecting Frontend And Backend**

**🌐 Step 13: Install whatwg-fetch in the Frontend**

**📁 Go to your frontend directory (where your React app is) and run:**

npm install --save whatwg-fetch

**✅ What this does:**

* Installs a **polyfill** for the Fetch API.
* Makes sure fetch() works even in **older browsers** or environments that don’t support it natively.

**🧠 Why use fetch()?**

fetch() is used to **make API calls** from your React app — for example:

fetch("http://localhost:8000/api/articles/react-basics")

.then((response) => response.json())

.then((data) => {

console.log(data); // Data from MongoDB

});

This is how your React frontend will **get data from Express**, and eventually **display articles, comments, etc.**