Ex. No.: 06	Take The Data Out Of The Views And Make The Smater
Date : 28/08/2024	

Aim:

To create a simple Express.js server that can handle GET and POST requests and return specific JSON responses.

Procedure:

1. Set Up the Project:

- Install Node.js and npm if not already installed on your system.
- Create a new folder for the project and open it in Visual Studio Code.
- Create a file named server.js and add the provided code to this file.

2. Install Dependencies:

- Open the terminal in Visual Studio Code (*View > Terminal*).
- Initialize a new npm project and install Express by running:

```
bash

npm init -y

npm install express
```

3. *Write Server Code*:

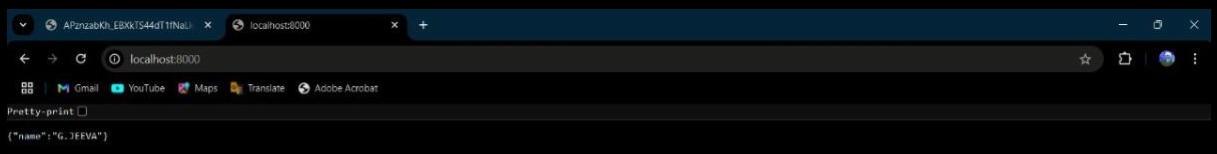
- In server.js, import Express and create an app instance to handle incoming requests.
- Use express.urlencoded({ extended: true }) as middleware to parse URL-encoded data in POST requests.
- Define a *GET* request handler on the root route (/) that responds with JSON data containing name: "G.JEEVA".
- Define a *POST* request handler on /get-data that responds with JSON data containing status: "done".

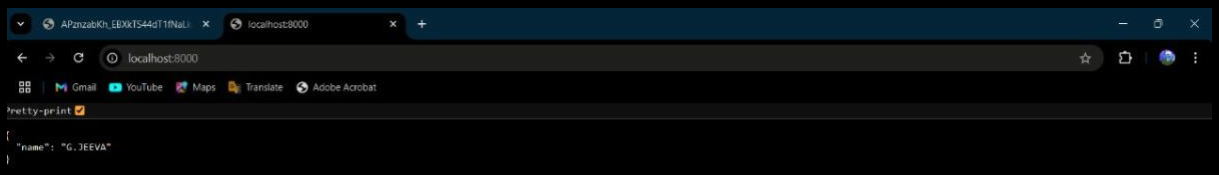
Source Code:

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

const app = express();

const port = 8000;
```

```
// Middleware to parse URL-encoded data
app.use(express.urlencoded({ extended: true }));

// GET request handler
app.get("/", (req, res) => {
    res.json({ name: "G.JEEVA" });
});

// POST request handler
app.post("/get-data", (req, res) => {
    res.json({ status: "done" });
});

// Start the server
app.listen(port, () => {
    console.log(App Running on port ${port});
});
```

Output:





Result:

The Express.js server is successfully set up and running. It responds to GET requests at the root URL with JSON data containing the user's name and to POST requests at /get-data with a JSON status confirmation. This demonstrates a basic understanding of setting up an Express server, defining routes, and handling HTTP requests and responses.