Experiment 5

Aim: To set up a basic Node.js server using Express and create simple HTTP endpoints to handle requests and responses.

Theory:

1. What is Node.js?

Node.js is a JavaScript runtime built on Chrome's V8 engine. It allows developers to run JavaScript on the server side to create scalable network applications.

2. What is Express.js?

Express.js is a lightweight and flexible web application framework for Node.js that simplifies server-side development by providing built-in functionalities like routing and middleware support.

3. Key Concepts:

- **Setting up a Node.js Project:**
  - Install Node.js.
  - Initialize a project using `npm init`.
  - Install Express using `npm install express`.

- **Creating Routes in Express:**
  - Express provides methods like `app.get()`, `app.post()`, `app.put()`, and `app.delete()` to define routes that handle different types of HTTP requests.

- **Handling HTTP Requests and Responses:**
  - A **request** is sent by the client to the server.
  - A **response** is sent back by the server to the client.
  - JSON data can be exchanged between the client and server.

- **Middleware in Express:**
  - Middleware functions process requests before sending responses.
  - Examples: `express.json()` for parsing JSON, logging, and error handling.

- **Error Handling in Express:**

- Express provides a mechanism to handle errors globally and send appropriate responses when something goes wrong.

Procedure:

Step 1: Setting Up the Node.js Project

1. Install **Node.js** from https://nodejs.org/.

   Open a terminal and create a project folder:
   ```
   mkdir node-server

   cd node-server
   ```

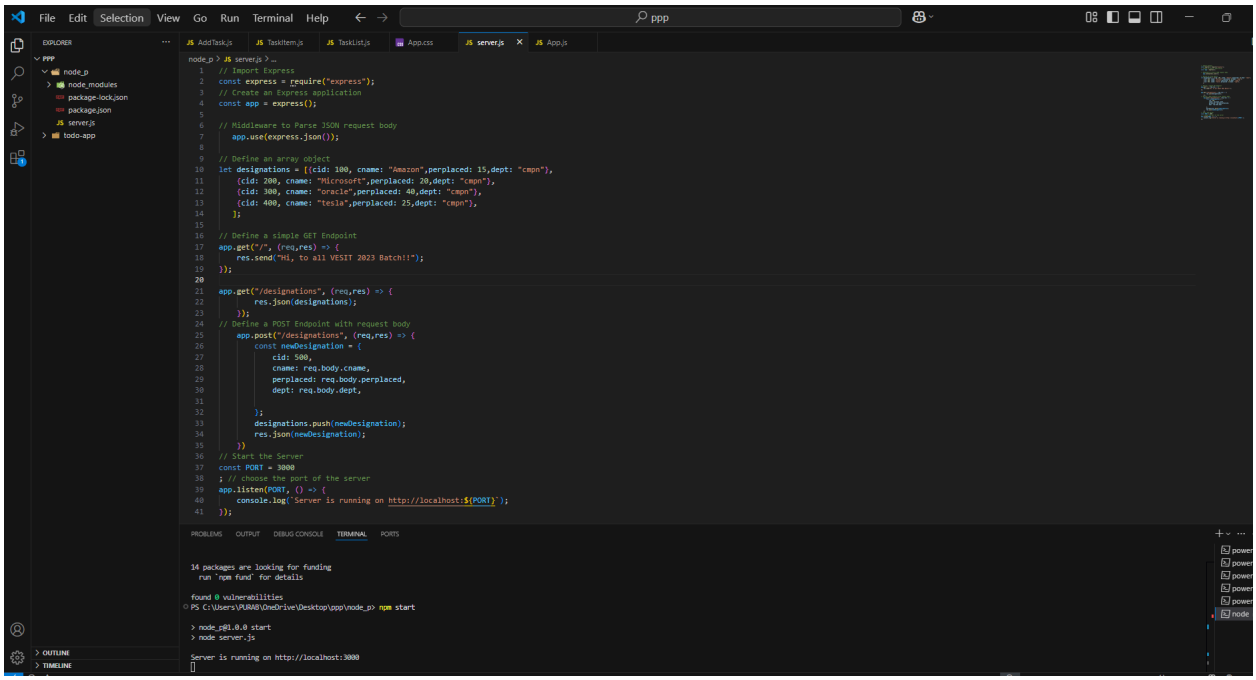
2. Initialize the project with:
   ```
   npm init -y
   ```
3. Install Express:
   ```
   npm install express
   ```

Step 2: Create and Configure the Server

Create a new file **server.js** and add the following code:
Code :

```
// Import Express
const express = require("express");
// Create an Express application
const app = express();

// Middleware to Parse JSON request body
  app.use(express.json());

// Define an array object
let designations = [{cid: 100, cname:
"Amazon",perplaced: 15,dept: "cmpn"},
  {cid: 200, cname: "Microsoft",perplaced:
20,dept: "cmpn"},
  {cid: 300, cname: "oracle",perplaced:
40,dept: "cmpn"},
  {cid: 400, cname: "tesla",perplaced:
25,dept: "cmpn"},
  ];

// Define a simple GET Endpoint
app.get("/", (req,res) => {
  res.send("Hi, to all VESIT 2023 Batch!!");
});

app.get("/designations", (req,res) => {
      res.json(designations);
  });
// Define a POST Endpoint with request
body
  app.post("/designations", (req,res) => {
      const newDesignation = {
```

```
    cid: 500,
    cname: req.body.cname,
    perplaced: req.body.perplaced,
    dept: req.body.dept,

};
designations.push(newDesignation);
res.json(newDesignation);
    })
// Start the Server
const PORT = 3000
; // choose the port of the server
app.listen(PORT, () => {
    console.log(Server is running on
http://localhost:${PORT});
});
```



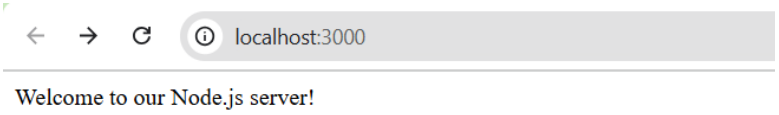
Step 3: Running the Server

Open the terminal and start the server:

```
node server.js
```

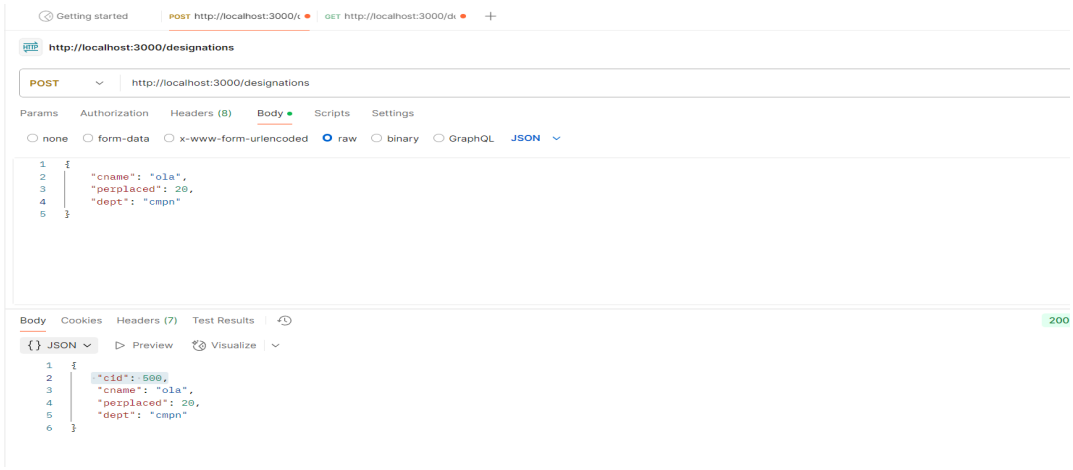
Step 4: Testing the Server

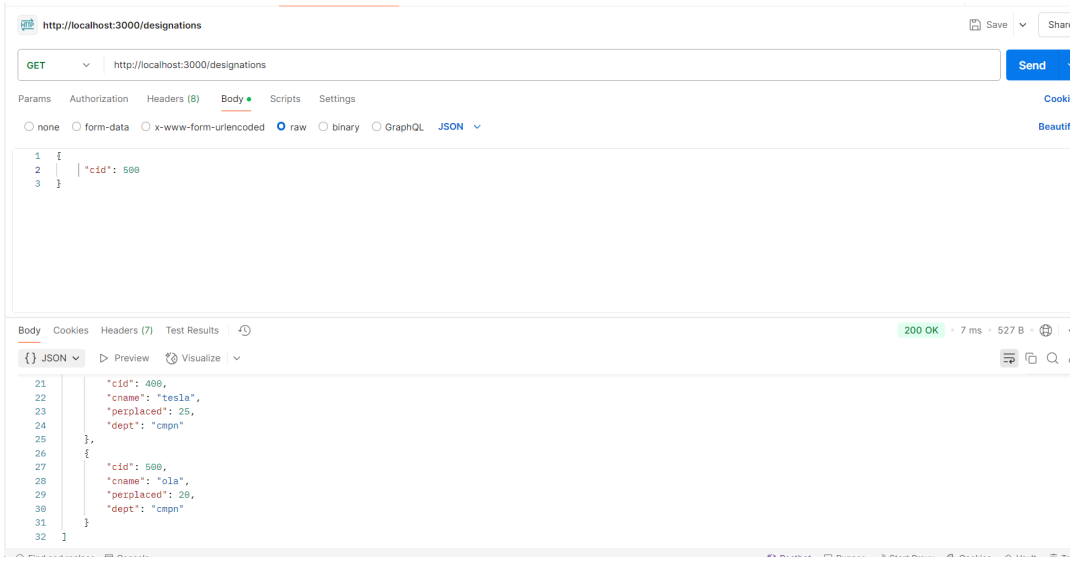
- **Open a browser** and visit:
    1. `http://localhost:3000/`

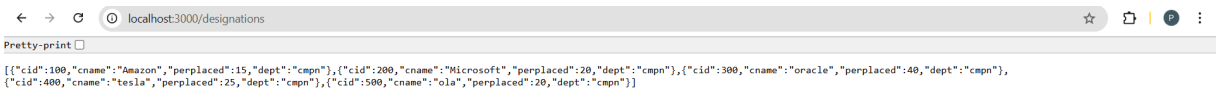


- **Testing with Postman:**

    1. Open **Postman**.
    2. Set request type to **POST**.
    3. Enter URL: `http://localhost:3000/designation`.







Conclusion:

In this experiment, we successfully set up a Node.js server using Express, created routes to handle GET and POST requests, implemented middleware for JSON parsing, and added error handling. By following these steps, we learned how to build a basic API that can communicate with clients, demonstrating the fundamental principles of server-side development with Node.js and Express.