EXPERIMENT NO. 5

AIM: Set up a basic Node.js server and create simple HTTP endpoints to handle requests and responses.

Exercises:
- Establishing a project with Node.js.
- Express route creation.
- Managing HTTP queries and answers.
- The handling of errors and middleware

THEORY:

- **Introduction to Express.**
  Express is a Node.js web application framework that simplifies and expedites the process of developing servers and APIs. For managing HTTP requests like GET, POST, PUT, and DELETE, it offers ready-to-use functionalities. Express offers versatility for both small and large projects and is both lightweight and powerful enough for large-scale applications. It streamlines routing, making it simple for developers to design distinct routes for various activities. Express's middleware support aids with error handling, data parsing, authentication, and logging. Development is accelerated by its extensive ecosystem of third-party libraries. Express is favored overall due to its ability to save time, simplify code, and scale for contemporary web applications.

- **Benefits of Creating Routes with Express Framework.**
  1. Modularity: Helps organize code by separating different functionalities into routes.
  2. Readability: Clear structure makes the code easier to read and manage.
  3. Dynamic Routing: Supports parameters and query strings for flexible URLs.
  4. Middleware Integration: Easily adds features like logging, authentication, and error handling.
  5. Scalability: Routes can be expanded as the project grows.
  6. Third-Party Support: Large ecosystem of libraries to enhance route functionalities.
  7. Faster Development: Reduces boilerplate code, speeding up the development process.

- **Uses of Middleware in Express Framework.**
  1. Logging: Tracks and records incoming requests for debugging and monitoring.
  2. Authentication: Validates users before allowing access to specific routes.
  3. Error Handling: Catches and manages errors without crashing the server.
  4. Parsing Data: Processes JSON, form data, or URL-encoded data in requests.
  5. Security: Adds headers and protects against common vulnerabilities.
  6. Serving Static Files: Delivers images, CSS, and JavaScript files to the client.

- **Steps to Handle HTTP Requests and Responses**
  1. Create a Server: Initialize a Node.js project and import Express.
  2. Define Routes: Set up routes for different HTTP methods like GET, POST, etc.
  3. Apply Middleware: Use middleware for parsing, logging, and security.
  4. Process Request: Handle client data, perform logic, or query the database.
  5. Send Response: Return appropriate responses in JSON, HTML, or text format.
  6. Handle Errors: Implement error-handling middleware to manage exceptions.

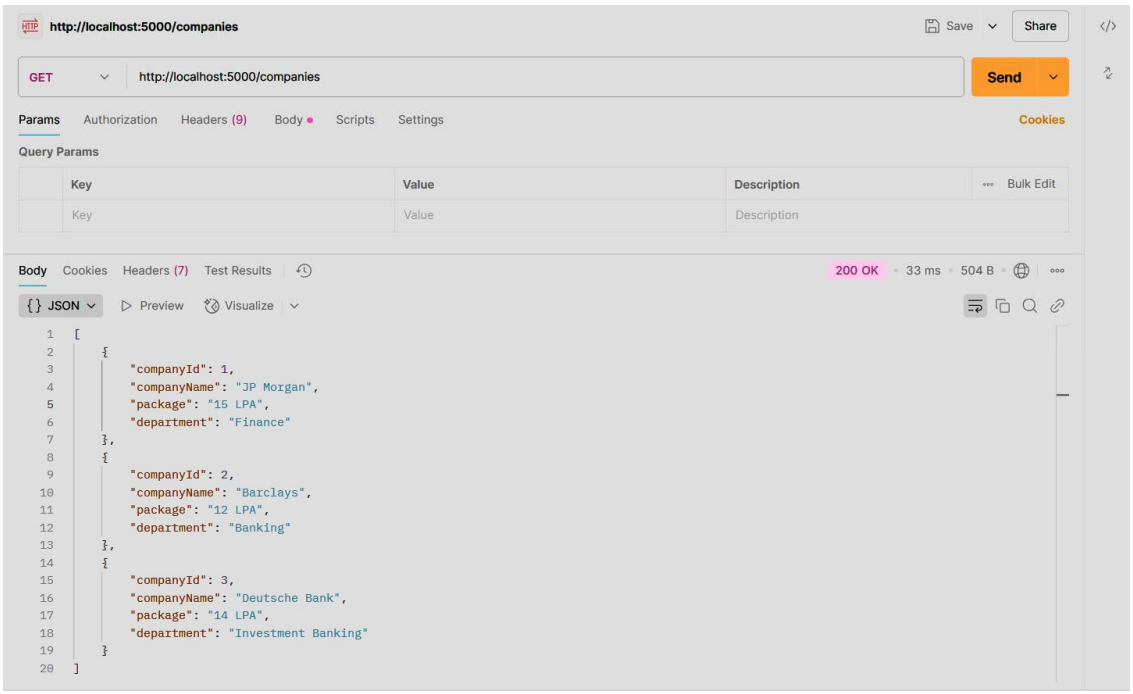
- **Postman API Development Tool.**
- A popular tool for developing APIs, Postman aids developers in creating, testing, and debugging APIs quickly. Without knowing any code, users may build and submit HTTP requests like GET, POST, PUT, and DELETE using its user-friendly interface. Postman is adaptable for a range of applications since it accepts multiple data types, including JSON, XML, and form data. Collaboration tools that let team members share workspaces and collections, as well as automated testing where developers may write scripts to evaluate replies, are some of its primary features. Additionally, Postman allows customers to effortlessly manage several settings, including development, testing, and production, and track API performance. All things considered, it makes testing APIs easier and helps guarantee that they function as intended prior to deployment.

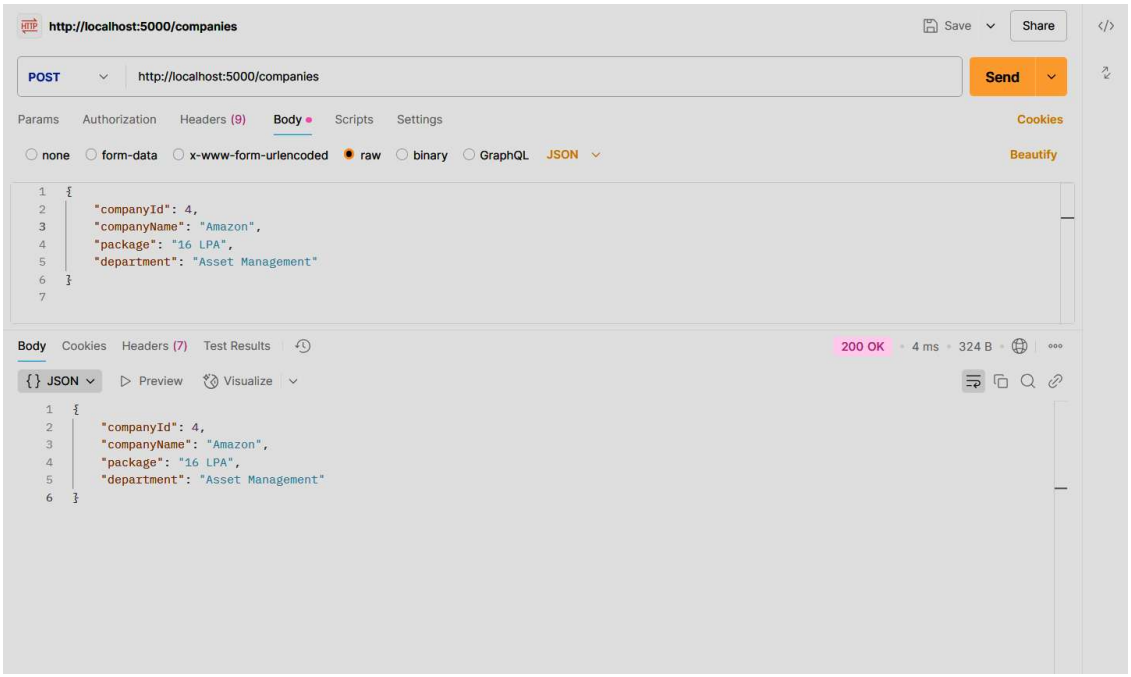
CODE

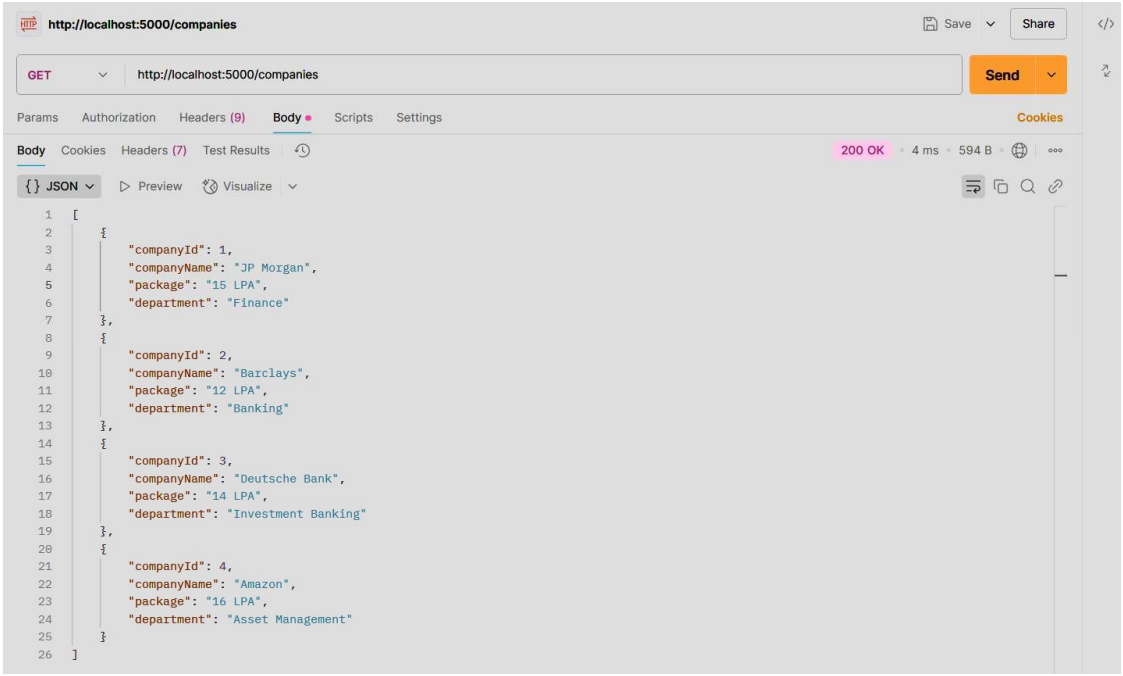
```
// Import Express
const express = require("express");
// Create an Express application
const app = express();

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

// Define an array object with company details let
companies = [
  {
    companyId: 1, companyName:
    "JP Morgan", package: "15
    LPA", department: "Finance"
  },
  {
    companyId: 2, companyName:
    "Barclays", package: "12 LPA",
    department: "Banking"
  },
  {
    companyId: 3,
    companyName: "Deutsche Bank", package:
    "14 LPA",
    department: "Investment Banking"
  }
```

```
];
// Define a simple GET Endpoint app.get("/",

(req, res) => {

    res.send("Hi, to all VESIT 2023 Batch!!");

});
// Endpoint to fetch all companies

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

    res.json(companies);

});
// Define a POST Endpoint with request body to add a new company

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

    const { companyId, companyName, package, department } = req.body;
    const newCompany = {

        companyId,

        companyName,

        package,  department

    };
    companies.push(newCompany); res.json(newCompan

})
;


// Start the Server

 const PORT = 5000; // choose the port of the serv
app.listen(PORT, () => {
console.log(`Server is running on http://localhost:${PORT}`);
});
```

OUTPUT


http://localhost:5000/companies
Save
Share
GET
http://localhost:5000/companies
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
33 ms
504 B
JSON
Preview
Visualize
[
{
"companyId": 1,
"companyName": "JP Morgan",
"package": "15 LPA",
"department": "Finance"
},
{
"companyId": 2,
"companyName": "Barclays",
"package": "12 LPA",
"department": "Banking"
},
{
"companyId": 3,
"companyName": "Deutsche Bank",
"package": "14 LPA",
"department": "Investment Banking"
}
]


http://localhost:5000/companies
Save
Share
POST
http://localhost:5000/companies
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"companyId": 4,
"companyName": "Amazon",
"package": "16 LPA",
"department": "Asset Management"
}
Body
Cookies
Headers (7)
Test Results
200 OK
4 ms
324 B
JSON
Preview
Visualize
{
"companyId": 4,
"companyName": "Amazon",
"package": "16 LPA",
"department": "Asset Management"
}



CONCLUSION

Express is a powerful and easy-to-use framework for building web servers in Node.js. It simplifies handling HTTP requests, routing, and middleware, making development faster and more organized. With features like modular routing and middleware support, it helps manage large projects efficiently.

Tools like Postman further make it easy to test and debug APIs before deployment. Overall, Express is a great choice for creating scalable and reliable web applications.