

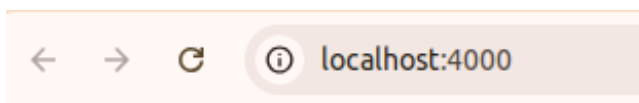
ExpressJS Assignment Report

Overview

This document outlines the steps I followed to complete an Express.js API project. The task was to create an Express server that handles both GET and POST requests, with the GET API returning query and path parameters, and the POST API accepting and saving JSON data to a file.

Step 1: Create an Express Server

- **Action:** Initialize the project and create an Express server.
- **Tools Used:** Node.js, npm, Express.
- **Description:** The project was initialized using `npm init -y`, and Express was installed to handle HTTP requests. I created the server to listen on port 4000 and set up middleware to handle incoming JSON data.
- **Commands:**
 - ❖ `npm init -y`
 - ❖ `npm install express`
- **Code Snippet for Server Setup:**



Welcome to the Express server!

```
const express = require('express');
const fs = require('fs');
const app = express();
const port = 4000;

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

Step 2: Create a GET API That Returns Query and Path Parameters

- **Action:** Implement a GET API that returns both path (id) and query (value) parameters.
- **Tools Used:** Express.js.
- **Description:** I created a GET API at the endpoint `/api/:id`. The path parameter (id) is accessed via `req.params.id`, and the query parameter (value) is accessed via `req.query.value`. Both parameters are returned as part of a JSON response.

- **Code Snippet for GET API:**

```
// Welcome message
app.get('/', (req, res) => {
  res.send('Welcome to the Express server!');
});

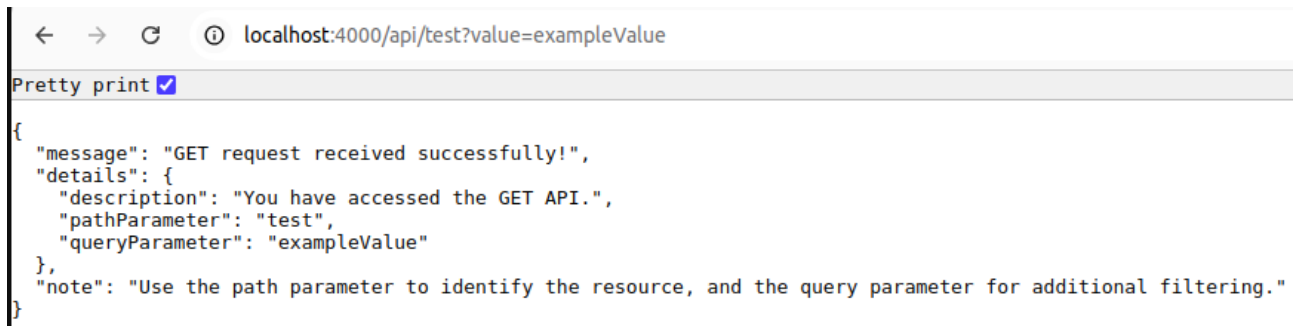
// GET API
app.get('/api/:id', (req, res) => {
  const pathParam = req.params.id; // Path parameter
  const queryParams = req.query.value; // Query parameter

  res.json({
    message: "GET request received successfully!",
    details: {
      description: "You have accessed the GET API.",
      pathParameter: pathParam,
      queryParameter: queryParams
    },
    note: "Use the path parameter to identify the resource, and the query parameter for additional filtering."
  });
});
```

- **GET Request:**

http://localhost:4000/api/test?value=exampleValue

- **Response:**



```
{
  "message": "GET request received successfully!",
  "details": {
    "description": "You have accessed the GET API.",
    "pathParameter": "test",
    "queryParameter": "exampleValue"
  },
  "note": "Use the path parameter to identify the resource, and the query parameter for additional filtering."
}
```

Step 3: Create a POST API to Accept and Save JSON Data

- **Action:** Implement a POST API that accepts incoming JSON data, saves it to a file (data.json), and returns the updated array of data.
- **Tools Used:** Express.js, fs (File System) module for reading and writing to files.
- **Description:** The POST API accepts a JSON object in the request body, appends it to existing data stored in a data.json file, and returns the updated data. The file is read using fs.readFile(), and new data is written using fs.writeFile().

- **Code Snippet for POST API:**

```
// POST API
app.post('/api/data', (req, res) => {
  const data = req.body; // Incoming data
  fs.readFile('data.json', (err, fileData) => {
    if (err) {
      console.error('Error reading file:', err);
      return res.status(500).json({ message: 'Error reading data file' });
    }
    // Parse the existing data or initialize an empty array
    let jsonData;
    try {
      jsonData = fileData.length > 0 ? JSON.parse(fileData) : [];
    } catch (parseError) {
      console.error('Error parsing JSON data:', parseError);
      return res.status(500).json({ message: 'Error parsing existing data' });
    }
    // Add the new data
    jsonData.push(data);
    fs.writeFile('data.json', JSON.stringify(jsonData, null, 2), (err) => {
      if (err) {
        console.error('Error writing file:', err);
        return res.status(500).json({ message: 'Error saving data' });
      }
      console.log('Data successfully saved:', jsonData);
      res.json({
        message: 'POST request successful, data saved to file',
        data: jsonData,
      });
    });
  });
});
```

- **POST Request:**

curl -X POST http://localhost:4000/api/data -H "Content-Type: application/json" -d
'{"name": "Elvicia Pinto", "age": 22, "email": "elviciampinto@gmail.com", "occupation":
"Developer"}'

- **Response:**

```
{ } data.json > ...
1  [
2    {
3      "name": "Elvicia Pinto",
4      "age": 22,
5      "email": "elviciampinto@gmail.com",
6      "occupation": "Developer"
7    }
8  ]
```

Step 4: Start the Server

- **Action:** Start the server to listen for incoming requests.
- **Tools Used:** Node.js.
- **Description:** I started the server using `app.listen()` to listen for requests on port 4000. The server logs a message to the console indicating it's running successfully.
- **Code Snippet for Starting the Server:**

```
// Start the server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

- **Response:**



Welcome to the Express server!

Testing

- **GET Request Test:** Accessed the GET API at `http://localhost:4000/api/test?value=exampleValue` and confirmed that both the path and query parameters were correctly returned in the response.
- **POST Request Test:** Sent a POST request using cURL, and confirmed that the JSON data was appended to `data.json` and the updated array was returned.

Conclusion

This project demonstrates the process of setting up an Express server with two APIs. The GET API retrieves both query and path parameters, and the POST API accepts JSON data, saves it to a file, and returns the updated data. Both APIs were successfully tested and are functioning as expected.