Mini Project Report - 14

Master of Computer Application – General
Semester – III

Sub: Web Technologies

Topic: REST APIs (middleware)
By
Name: SANDRA B
Reg no.: 24110222500001

Faculty Name: VEERA RAGHAV K

Faculty Signature: ________________

Department of Computer Application
Alliance University
Chandapura - Anekal Main Road, Anekal
Bengaluru - 562 106

August 2025

Mini Project-14: RESTful API Routes and Middleware for Car Data

1. Introduction

In the modern world of web development, the need for efficient communication between clients and servers has grown rapidly. With the increase in web and mobile applications, **RESTful APIs (Representational State Transfer APIs)** have become one of the most widely used approaches for enabling this communication. REST APIs provide a standardized way for data to be requested, transferred, and manipulated over the web using simple HTTP methods such as **GET, POST, PUT, and DELETE**.

The project **"Car Data Management using RESTful API, Routes, and Middleware"** demonstrates the creation of a local server using **Node.js** and **Express.js**. The main purpose of the server is to handle client requests, process data, and return appropriate responses. In this project, sample car data is used to show how a REST API can be implemented with middleware support.

This project will help students and beginners understand the concepts of:

- REST architecture
- API routes
- Middleware in Express.js
- Handling JSON data in Node.js applications

By the end of this report, the reader will clearly understand how to set up a RESTful API using Express, send and receive data in JSON format, and utilize middleware effectively.

2. Objective

The key objectives of this project are as follows:

1. To design and implement a **local server** using Node.js and Express.js.
2. To demonstrate the concept of **middleware** for processing JSON data.
3. To define **API routes** that allow the client to interact with the server.
4. To build a sample **RESTful API for car data management**.
5. To understand how to handle client requests such as fetching and sending data.
6. To explore the **advantages and future scope** of REST APIs in real-world applications.

In summary, the objective is to learn and showcase how REST APIs function in practice, focusing on the **Car Dataset API**.

3. Tools and Technologies Used

For the development of this project, the following tools and technologies were used:

1. **Node.js**
   - A runtime environment that allows JavaScript to be executed outside the browser.
   - Helps in building scalable and fast server-side applications.
2. **Express.js**
   - A lightweight web application framework for Node.js.
   - Provides features for handling routes, middleware, and HTTP requests.
3. **JavaScript (ES6)**
   - The core programming language used to implement the logic of the project.
4. **Middleware (Express.json)**
   - Middleware is used for parsing incoming JSON requests.
5. **REST Architecture**
   - REST (Representational State Transfer) defines principles for building APIs that are stateless, scalable, and easy to use.
6. **Localhost (Port 3000)**
   - The server is hosted locally on port 3000.
7. **Text Editor / IDE**
   - VS Code is used to write and test the code.
8. **Postman / Browser**
   - Tools used to test the APIs and visualize the responses.

4. Description of Code

The code can be divided into different sections:

(a) Importing Express

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

const app = express();
```

Here, the Express module is imported and an application object is created. This app object is used to define routes and middleware.

(b) Middleware

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

This line adds middleware to parse incoming JSON data. Middleware acts as a bridge between the request and the response cycle.

(c) GET Request for Car Data

```
app.get("/api/users", (req, res) => {
  const cars = [
    { carid: 101, carname: "Honda Civic", carcolor: "Red", carprice: 1200000 },
    { carid: 102, carname: "Toyota Fortuner", carcolor: "Black", carprice: 3500000 },
    { carid: 103, carname: "Hyundai i20", carcolor: "White", carprice: 900000 }
  ];
  res.send(cars);
});
```

This route handles the GET request to /api/users. It responds with a list of cars in JSON format.

(d) POST Request for Adding Cars

```
app.get("/api/users", (req,res)=>{
  const newCars = req.body();
  res.send({
    message:"User are Created!!s",
    cars:newCars
  })
});
```

Here, the server is designed to handle incoming car data from the client using req.body(). However, it should ideally be a POST request instead of GET.

(e) Server Listening

```
app.listen(3000, () => {
  console.log("localhost:3000/api/users");
});
```

This line starts the server on port 3000 and logs a message indicating the endpoint URL.

---

5. Working of the Project

The working can be explained step by step:

1. **Start the Server**
   Run the command node index.js (or your file name). This launches the local server at http://localhost:3000/.
2. **Middleware in Action**
   When a client sends JSON data, the middleware express.json() processes and makes it accessible through req.body.
3. **GET Request (Retrieve Cars)**
   A client/browser/Postman can send a GET request to /api/users.
   - The server responds with an array of car objects.
   - Each object contains car ID, name, color, and price.
4. **POST Request (Add Cars)**
   A client sends a POST request with new car data in JSON format.
   - The server accepts this data and sends back a confirmation message.
5. **Response**
   The server returns data in JSON, ensuring compatibility with front-end frameworks (React, Angular, etc.) and mobile apps.

---

6. Features

- Simple **REST API** implementation.
- Ability to **fetch all car details** using GET.
- Middleware support for handling JSON.
- Easy to extend and add more routes (PUT, DELETE).
- Runs locally and can be tested with browsers or Postman.
- Clear demonstration of **client-server communication**.

---

7. Output

Example 1: GET Request

URL: http://localhost:3000/api/users
Response:

```
[
  { "carid": 101, "carname": "Honda Civic", "carcolor": "Red", "carprice": 1200000 },
  { "carid": 102, "carname": "Toyota Fortuner", "carcolor": "Black", "carprice": 3500000 },
  { "carid": 103, "carname": "Hyundai i20", "carcolor": "White", "carprice": 900000 }
]
```

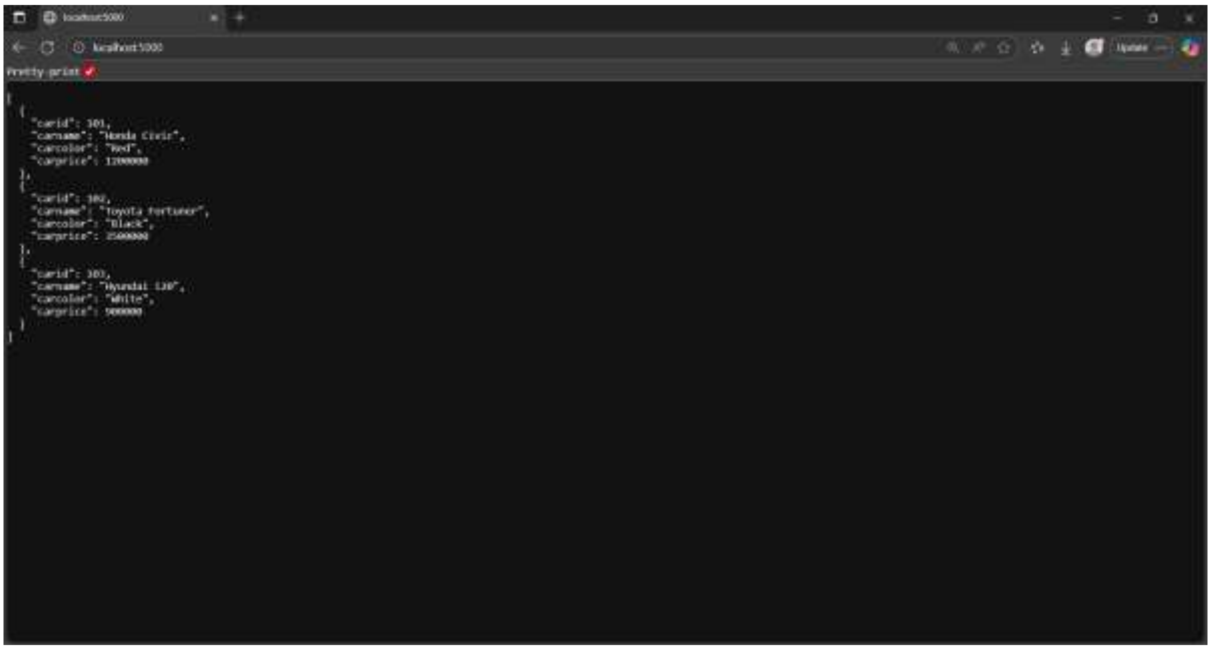
Example 2: POST Request

Request (Body):

```
{ "carid": 104, "carname": "Kia Seltos", "carcolor": "Blue", "carprice": 1500000 }
```

Response:

```
{
  "message": "User are Created!!s",
  "cars": { "carid": 104, "carname": "Kia Seltos", "carcolor": "Blue", "carprice": 1500000 }
}
```



```
[
  {
    "carid": 101,
    "carname": "Honda Civic",
    "carcolor": "Red",
    "carprice": 1200000
  },
  {
    "carid": 102,
    "carname": "Toyota Fortuner",
    "carcolor": "Black",
    "carprice": 2500000
  },
  {
    "carid": 103,
    "carname": "Hyundai i20",
    "carcolor": "White",
    "carprice": 900000
  }
]
```

8. Advantages

1. **Simplicity** – REST APIs are easy to understand and implement.
2. **Scalability** – The server can handle more data and endpoints easily.
3. **Flexibility** – Any client (web, mobile) can consume the API.
4. **JSON Support** – Widely accepted data format in modern applications.
5. **Middleware** – Helps in handling and validating requests.
6. **Extensibility** – Can add authentication, error handling, or a database later.

---

9. Future Scope

This mini-project can be expanded into a complete car management system with:

- **Database Integration** – Use MongoDB or MySQL to store car details permanently.
- **CRUD Operations** – Add PUT (update) and DELETE routes.
- **Authentication & Authorization** – Secure API with JWT or OAuth.
- **Frontend Integration** – Connect with React, Angular, or Vue.
- **Error Handling** – Add middleware for error detection and reporting.
- **Deployment** – Host on platforms like Heroku, Vercel, or AWS.

---

10. How to Run the Project

To execute this project successfully, the following steps were followed:

Step 1: Install Node.js and NPM

Ensure Node.js is installed on the system. It automatically comes with NPM (Node Package Manager), which is used to manage dependencies.

Check versions:

```
node -v
```

```
npm -v
```

Step 2: Initialize the Project

Navigate to the project folder and initialize with default values:

```
npm init -y
```

This creates a package.json file that stores metadata about the project. After initialization, the following snippet was generated:

```
{
  "dependencies": {
    "express": "^5.1.0",
    "prompt-sync": "^4.2.0"
  },
  "name": "sandra-b",
  "version": "1.0.0",
  "main": "index.js"
}
```

Step 3: Install Express

Express.js was installed as the main dependency for creating the RESTful API server:

```
npm install express
```

The installation completed successfully with no vulnerabilities.

Step 4: Create the Server File

Inside the project folder, a new JavaScript file was created (server.js or sever.js) that contains the Express code.

Step 5: Running the Server

Navigate to the project directory:

```
cd C:\Users\SANDRA B\Documents\MCA Alliance cllg\Sem 3\Web Technologies\Mini Projects\Sever\my-express
```

Initially, there was an error because the filename was typed incorrectly:

```
node server.js
```

```
Error: Cannot find module '...server.js'
```

This happened because the file was named sever.js instead of server.js.

After correcting the command:

```
node sever.js
```

The server started successfully with the message:

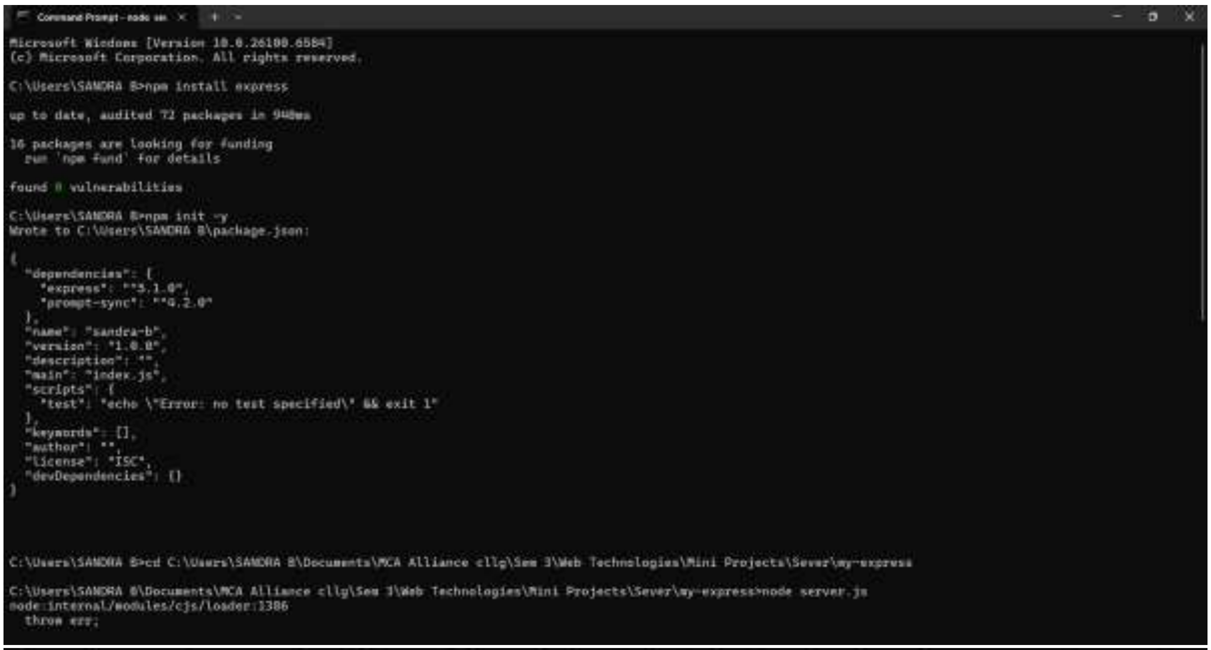
```
Sever is running successfully using localhost:5000
```

Step 6: Testing the Endpoints

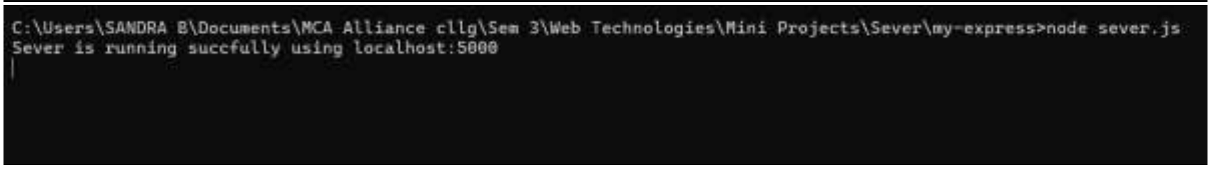
Open a browser or Postman and test the following:

- GET Request:
  http://localhost:5000/api/users

- POST Request:
  Send JSON data to http://localhost:5000/api/users



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SANDRA B>npm install express

up to date, audited 72 packages in 940ms

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\SANDRA B>npm init -y
Wrote to C:\Users\SANDRA B\package.json:

{
  "dependencies": {
    "express": "^5.1.0",
    "prompt-sync": "^4.2.0"
  },
  "name": "sandra-b",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {}
}



C:\Users\SANDRA B>cd C:\Users\SANDRA B\Documents\MCA Alliance cllg\Sem 3\Web Technologies\Mini Projects\Sever\my-express

C:\Users\SANDRA B\Documents\MCA Alliance cllg\Sem 3\Web Technologies\Mini Projects\Sever\my-express>node server.js
node:internal/modules/cjs/loader:1386
  throw err;
```



```
C:\Users\SANDRA B\Documents\MCA Alliance cllg\Sem 3\Web Technologies\Mini Projects\Sever\my-express>node sever.js
Sever is running succfully using localhost:5000
```

11. Conclusion

This project serves as a **foundation for understanding REST APIs and middleware in Node.js applications**. By implementing car data management, we explored the basics of Express.js, routes, and JSON handling.

The project highlights the importance of middleware in processing client data, demonstrates how API endpoints can be created, and shows real-time interaction between client and server.

In the future, this system can evolve into a robust application with database connectivity, authentication, and advanced features. Overall, this project strengthens the understanding of **server-side development and RESTful services**, which are crucial in today's software industry.