

Titles

Project Name – Build a RESTful API using Node.js and Express

Build by – Shashikant Sahu

Batch – 1st March 2025

Table of Contents

1. Introduction
2. Creativity and Presentation
 - Technologies Used
 - code
 - Screenshot
3. GitHub Link

Introduction

This assignment demonstrates the creation of a RESTful API using Node.js and Express.js. The objective is to develop a server-side application that allows basic CRUD (Create, Read, Update, Delete) operations for managing user data. The project focuses on the core concepts of routing, middleware, HTTP methods, status codes, and error handling, using in-memory storage for simplicity.

The API includes endpoints for:

- Retrieving all users
- Retrieving a user by ID
- Adding a new user
- Updating an existing user
- Deleting a user

Creativity and Presentation

This assignment ensures that each responsibility is clearly separated, making the codebase easy to understand and maintain.

Technologies Used

To build and structure this RESTful API project efficiently, the following technologies were used:

- **Node.js**
A JavaScript runtime environment used to build the server-side application. It allows writing backend logic using JavaScript.
- **Express.js**
A fast and minimal web framework for Node.js that simplifies routing, handling HTTP requests, and middleware integration.
- **JavaScript (ES6+)**
The core programming language used throughout the project, leveraging modern syntax for cleaner and more efficient code.
- **ThunderClient (*API Testing Tool*)**
Used to test all API endpoints (GET, POST, PUT, DELETE) to ensure proper functionality, validation, and error handling.
- **VS Code (*Code Editor*)**
A powerful and customizable editor used to write and manage project files effectively.

Code

Index.js

```
const express = require("express");
const app = express();
const port = 3000;

// Middleware to parse JSON
app.use(express.json());

// Sample User Object Structure:
let users = [
  {
    id: "1",
    firstName: "Anshika",
    lastName: "Agarwal",
    hobby: "Teaching",
  },
  {
    id: "2",
    firstName: "shashikant",
    lastName: "sahu",
    hobby: "cricket",
  },
  {
    id: "3",
    firstName: "vivek",
    lastName: "kumar",
    hobby: "nothing",
  },
];

// Middleware for logging requests
app.use((req, res, next) => {
  res.on("finish", () => {
    console.log(`${req.method} ${req.originalUrl} - ${res.statusCode}`);
  });
});
```

```
    console.log("next middleware");
  });
  next();
});
```

// validation middleware

```
function validateUser(req, res, next) {
  const { firstName, lastName, hobby } = req.body;
  if (!firstName || !lastName || !hobby) {
    return res.status(400).json({ error: "All fields are required" });
  }
  next();
}
```

// GET /users – Fetch the list of all users.

```
app.get("/users", (req, res) => {
  res.status(200).json(users);
});
```

// GET /users/:id – Fetch details of a specific user by ID.

```
app.get("/users/:id", (req, res) => {
  const userId = req.params.id; //for dynamic routing.
  const user = users.find((user) => user.id === userId);
  if (!user) {
    return res.status(404).json({ error: "User not found with this id" });
  }
  res.status(200).json(user);
});
```

// POST /user – Add a new user.

```
app.post("/user", validateUser, (req, res) => {
  const lastId = users.length > 0 ? parseInt(users[users.length - 1].id) : 0;
  const newUser = {
    id: (lastId + 1).toString(),
    firstName: req.body.firstName,
    lastName: req.body.lastName,
    hobby: req.body.hobby,
  };
});
```

```
    users.push(newUser);  
    res.status(201).json(newUser);  
  });
```

// PUT /user/:id – Update details of an existing user.

```
app.put("/user/:id", validateUser, (req, res) => {  
  const userId = req.params.id;  
  const userIndex = users.findIndex((user) => user.id === userId);  
  if (userIndex === -1) {  
    return res.status(404).json({ error: "User not found with this id" });  
  }  
  users[userIndex] = { id: userId, ...req.body };  
  res.status(200).json(users[userIndex]);  
});
```

// DELETE /user/:id – Delete a user by ID.

```
app.delete("/user/:id", (req, res) => {  
  const userId = req.params.id;  
  const userIndex = users.findIndex((user) => user.id === userId);  
  if (userIndex === -1) {  
    return res.status(404).json({ error: "User not found with this id" });  
  }  
  const deletedUser = users.splice(userIndex, 1);  
  res.status(200).json(deletedUser[0]);  
});
```

//start server

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

Package.json

```
{
  "name": "buildingrestfulapi",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^5.1.0"
  }
}
```


Screenshot :-

GET- <http://localhost:3000/users>

The screenshot shows the VS Code interface with a REST client tab. The request is a GET to `http://localhost:3000/users`. The response is a 200 OK status with a JSON body containing user information.

Query: `http://localhost:3000/users`

Response: Status: 200 OK, Size: 74 Bytes, Time: 29 ms

```
1 [
2   {
3     "id": "1",
4     "firstName": "Anshika",
5     "lastName": "Agarwal",
6     "hobby": "Teaching"
7   }
8 ]
```

Terminal:

```
PS D:\Programming\Internshala\Project_Assignments\BuildingRESTfulAPI> node index.js
Server is running on http://localhost:3000
GET /users - 200
```

GET- <http://localhost:3000/users/2>

The screenshot shows the VS Code interface with a REST client tab. The request is a GET to `http://localhost:3000/users/2`. The response is a 200 OK status with a JSON body containing user information for ID 2.

Query: `http://localhost:3000/users/2`

Response: Status: 200 OK, Size: 71 Bytes, Time: 26 ms

```
1 {
2   "id": "2",
3   "firstName": "shashikant",
4   "lastName": "sahu",
5   "hobby": "cricket"
6 }
```

Terminal:

```
PS D:\Programming\Internshala\Project_Assignments\BuildingRESTfulAPI> node index.js
Server is running on http://localhost:3000
GET /users/2 - 200
next middleware
```

GET - http://localhost:3000/users/55

The screenshot shows the VS Code REST Client interface. The top tab is labeled 'localhost:5100'. The request is a GET to 'http://localhost:3000/users/55'. The status bar indicates 'Status: 404 Not Found', 'Size: 39 Bytes', and 'Time: 29 ms'. The response body is a JSON object:

```
{
  "error": "User not found with this id"
}
```

. The bottom terminal shows the command 'node index.js' and the output 'Server is running on http://localhost:3000', 'GET /users/55 - 404', and 'next middleware'.

POST - http://localhost:3000/user

The screenshot shows the VS Code REST Client interface. The top tab is labeled 'localhost:3000' and 'localhost:3000/users'. The request is a POST to 'http://localhost:3000/user'. The status bar indicates 'Status: 201 Created', 'Size: 67 Bytes', and 'Time: 61 ms'. The request body is a JSON object:

```
{
  "firstName": "vivek",
  "lastName": "kumar",
  "hobby": "nothing"
}
```

. The response body is a JSON object:

```
{
  "id": "4",
  "firstName": "vivek",
  "lastName": "kumar",
  "hobby": "nothing"
}
```

. The bottom terminal shows the command 'node index.js' and the output 'Server is running on http://localhost:3000', 'POST /user - 201', and 'next middleware'.

PUT - http://localhost:3000/user/1

The screenshot shows the VS Code REST Client interface. The top bar indicates the active tab is 'localhost:3000/user/1'. The request is a PUT method to 'http://localhost:3000/user/1'. The status is '200 OK', size is '67 Bytes', and time is '41 ms'. The response body is a JSON object: { "id": "1", "firstName": "ritik", "lastName": "kumar", "hobby": "dancing" }. The terminal at the bottom shows the following output:

```
GET /users - 200
next middleware
PUT /user/1 - 200
next middleware
```

PUT - http://localhost:3000/user/111

The screenshot shows the VS Code REST Client interface. The top bar indicates the active tab is 'localhost:3000/user/111'. The request is a PUT method to 'http://localhost:3000/user/111'. The status is '404 Not Found', size is '39 Bytes', and time is '59 ms'. The response body is a JSON object: { "error": "User not found with this id" }. The terminal at the bottom shows the following output:

```
PS D:\Programming\Internshala\Project_Assignments\BuildingRESTfulAPI> node index.js
Server is running on http://localhost:3000
PUT /user/111 - 404
next middleware
```

DELETE - http://localhost:3000/user/1

The screenshot shows the VS Code REST Client interface. The top bar indicates the active tab is 'localhost:3000/user/1'. The method is set to 'DELETE' and the URL is 'http://localhost:3000/user/1'. The 'Send' button is visible. The 'Query' tab is selected, showing 'Query Parameters' with a table with columns 'parameter' and 'value'. The 'Response' tab is also visible, showing the response body:

```
1 {
2   "id": "1",
3   "firstName": "Anshika",
4   "lastName": "Agarwal",
5   "hobby": "Teaching"
6 }
```

 The status bar at the bottom shows 'DELETE /user/1 - 200' and 'next: middleware'.

DELETE - http://localhost:3000/user/11

The screenshot shows the VS Code REST Client interface. The top bar indicates the active tab is 'localhost:3000/user/11'. The method is set to 'DELETE' and the URL is 'http://localhost:3000/user/11'. The 'Send' button is visible. The 'Query' tab is selected, showing 'Query Parameters' with a table with columns 'parameter' and 'value'. The 'Response' tab is also visible, showing the response body:

```
1 {
2   "error": "User not found with this id"
3 }
```

 The status bar at the bottom shows 'DELETE /user/11 - 404' and 'next: middleware'.