

# **Node Assignment 1:**

**Build a RESTful API using  
Node.js and Express**

**Submitted By :**  
*Chaitali Mahato*

Github: <https://github.com/chaitali9497/restful-api>

# PROJECT OVERVIEW :

This project involves developing a simple RESTful API using Node.js and Express for managing user information. The application allows performing basic CRUD operations such as creating, retrieving, updating, and deleting users. It demonstrates core backend development concepts including RESTful routing, middleware implementation, handling different HTTP methods, proper use of status codes, structured error handling, and in-memory data storage for managing user data efficiently.

## Technologies Used:

- *Node.js*
- *Express.js*
- *Thunder Client (for API testing)*
- *JavaScript*

## Project Setup

- Initialized a Node.js project using `npm init`.
- Installed Express framework.
- Created `server.js` as the main entry file.
- Configured middleware for JSON parsing, logging, and validation.

## API Endpoints Implemented:

<i>Method</i>	<i>Endpoint</i>	<i>Description</i>
<b>GET</b>	<code>/users</code>	<b>Fetch all users</b>
<b>GET</b>	<code>/users/:id</code>	<b>Fetch user by ID</b>
<b>POST</b>	<code>/user</code>	<b>Add a new user</b>
<b>PUT</b>	<code>/user/:id</code>	<b>Update an existing user</b>
<b>DELETE</b>	<code>/user/:id</code>	<b>Delete a user</b>

## ***Middleware Implementation***

### **1. Logging Middleware**

- Logs request method and URL for every API call.
- Helps in monitoring incoming requests.

### **2. Validation Middleware**

- Validates required fields (firstName, lastName, hobby) in POST and PUT requests.
- Returns 400 Bad Requests if any field is missing.

## ***Error Handling***

The API uses proper HTTP status codes and meaningful error messages:

<b><i>Scenario</i></b>	<b><i>Status Code</i></b>
<i>Successful Request</i>	200
<i>User Created</i>	201
<i>User Not Found</i>	404
<i>Invalid Input</i>	400

## Test Results :

### 1) GET all users

#### Request:

Method: GET

URL: http://localhost:3000/users

#### Response:

```
{  
  "id": "1",  
  "firstName": "Chaitali",  
  "lastName": "Mahato",  
  "hobby": "Coding"}  
}
```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/users
- Status:** 200 OK
- Size:** 72 Bytes
- Time:** 26 ms
- Response:**

```
1 [  
2 {  
3   "id": "1",  
4   "firstName": "Chaitali",  
5   "lastName": "Mahato",  
6   "hobby": "Coding"  
7 }  
8 ]
```

## 2) GET user by ID

Request:

Method: GET

URL: http://localhost:3000/users/1

Response:

```
{  
  "id": "1",  
  "firstName": "Chaitali",  
  "lastName": "Mahato",  
  "hobby": "Coding"  
}
```

The screenshot shows a REST client interface with two main sections: a left panel for sending requests and a right panel for viewing responses.

**Left Panel (Request):**

- Method: GET
- URL: http://localhost:3000/users/1
- Body tab is selected.
- Content Type: JSON
- JSON Content:

```
1
```

**Right Panel (Response):**

- Status: 200 OK
- Size: 70 Bytes
- Time: 12 ms
- Response tab is selected.
- Headers (6): [Listed but not detailed]
- Cookies: [Listed but not detailed]
- Results: [Listed but not detailed]
- Docs: [Listed but not detailed]
- JSON Response:

```
1  {  
2    "id": "1",  
3    "firstName": "Chaitali",  
4    "lastName": "Mahato",  
5    "hobby": "Coding"  
6  }
```

### 3) POST add user

#### Request:

Method: POST

URL: http://localhost:3000/user

Body:

```
{  
  "firstName": "Debashish",  
  "lastName": "Mahato",  
  "hobby": "Driving"  
}
```

#### Response:

```
{  
  "id": "1767204850064",  
  "firstName": "Debashish",  
  "lastName": "Mahato",  
  "hobby": "Coding"  
}
```

The screenshot shows the Postman application interface. On the left, the 'Body' tab is selected, showing a JSON content block with the following code:

```
1 {  
2   "firstName": "Debashish",  
3   "lastName": "Mahato",  
4   "hobby": "Driving"  
5 }
```

On the right, the response section displays the following details:

- Status: 201 Created
- Size: 84 Bytes
- Time: 4 ms

The response body is shown as:

```
1 {  
2   "id": "1767204850064",  
3   "firstName": "Debashish",  
4   "lastName": "Mahato",  
5   "hobby": "Driving"  
6 }
```

## 4) PUT update user

### Request:

Method: PUT

URL: http://localhost:3000/user/1767204850064

Body:

```
{  
  "firstName": "Debashish",  
  "lastName": "Mahato",  
  "hobby": "Reading"  
}
```

### Response:

```
{  
  "id": "1767204850064",  
  "firstName": "Debashish",  
  "lastName": "Mahato",  
  "hobby": "Reading"  
}
```

The screenshot shows the Postman application interface. On the left, the request details are set: Method is PUT, URL is http://localhost:3000/user/1767204850064, and the Body tab is selected. The body content is a JSON object with fields id, firstName, lastName, and hobby. On the right, the response details are shown: Status is 200 OK, Size is 84 Bytes, and Time is 4 ms. The Response tab displays the returned JSON object, which matches the sent body.

Header	Value
Status	200 OK
Size	84 Bytes
Time	4 ms

Section	Value
Response	{ "id": "1767204850064", "firstName": "Debashish", "lastName": "Mahato", "hobby": "Reading" }
Headers	Content-Type: application/json
Cookies	
Results	
Docs	

## 5) DELETE user

### Request:

Method: DELETE

URL: http://localhost:3000/user/1767204850064

### Response:

```
{  
  "message" : "User deleted successfully"  
}
```

The screenshot shows the Postman interface. On the left, the request method is set to 'DELETE' and the URL is 'http://localhost:3000/user/1767204850064'. The 'Body' tab is selected. On the right, the response details are shown: Status: 200 OK, Size: 39 Bytes, Time: 3 ms. The 'Response' tab is selected, displaying the JSON response: { "message": "User deleted successfully" }.

----- THANK YOU -----