

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

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:

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

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 displays a REST client interface with a dark theme. The top bar shows the method 'GET' and the URL 'http://localhost:3000/users'. The 'Send' button is highlighted. Below the top bar, the 'Query' tab is selected, showing 'Query Parameters' with a table for parameters. The 'Response' tab is also visible, showing the status '200 OK', size '72 Bytes', and time '26 ms'. The response body is a JSON object:

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

parameter	value
-----------	-------

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

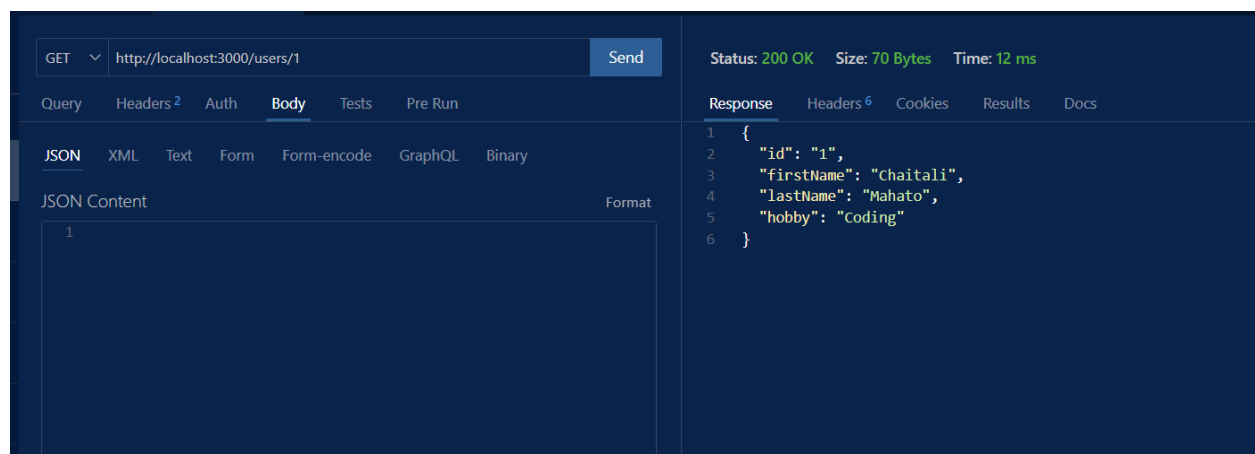
Request:

Method: GET

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

Response:

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



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 displays a REST client interface with a dark theme. The top bar shows the method 'POST' and the URL 'http://localhost:3000/user'. The 'Send' button is highlighted. Below the top bar, the 'Body' tab is selected, showing the request body in JSON format:

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

. The right panel shows the response status '201 Created', size '84 Bytes', and time '4 ms'. The 'Response' tab is selected, displaying the response body in JSON format:

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

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 displays a REST client interface with a PUT request to `http://localhost:3000/user/1767204850064`. The request body is a JSON object: `{ "firstName": "Debashish", "lastName": "Mahato", "hobby": "Reading" }`. The response status is `200 OK` with a size of `84 Bytes` and a time of `4 ms`. The response body is a JSON object: `{ "id": "1767204850064", "firstName": "Debashish", "lastName": "Mahato", "hobby": "Reading" }`.

PUT	http://localhost:3000/user/1767204850064	Send
Query	Headers ²	Auth
Body ¹	Tests	Pre Run
JSON	XML	Text
Form	Form-encode	GraphQL
Binary		
JSON Content		Format
1	{	
2	"firstName": "Debashish",	
3	"lastName": "Mahato",	
4	"hobby": "Reading"	
5	}	

Status: 200 OK	Size: 84 Bytes	Time: 4 ms
Response	Headers ⁶	Cookies
Results		Docs
1	{	
2	"id": "1767204850064",	
3	"firstName": "Debashish",	
4	"lastName": "Mahato",	
5	"hobby": "Reading"	
6	}	

5) DELETE user

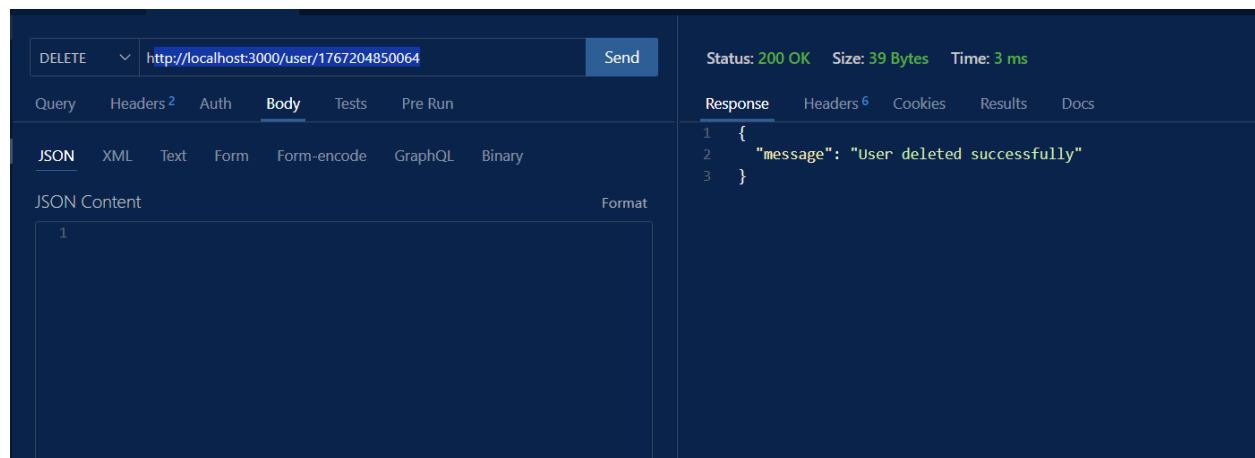
Request:

Method: DELETE

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

Response:

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



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