

Assignment: Build a RESTful API using Node.js and Express (100 marks)

1. Project Overview

This project is a simple REST API built using Express.js that allows us to **manage user information** stored in an in-memory JavaScript array.

Using this API, we can perform the following operations:

- View all users
- View a specific user by ID
- Add a new user
- Update an existing user
- Delete a user

All these operations are performed through HTTP methods like **GET, POST, PUT, and DELETE**, and each request returns meaningful responses through proper status codes.

2. Middleware Overview

Middlewares help us interact with every incoming request before it reaches the actual route.

a) Logging Middleware

Whenever we hit an API endpoint, this middleware records useful information such as:

- Which HTTP method was used (GET/POST/PUT/DELETE)
- Which URL was accessed
- What response status code was sent back

Why is it useful?

It helps us debug easily and monitor every request for performance and errors.

b) Validation Middleware

This middleware checks whether the request body contains **all required fields** before adding or updating a user.

- For adding a user, fields like **id, first name, last name, and hobby must be present**.
- For updating a user, fields like **first name, last name, and hobby must not be empty**.

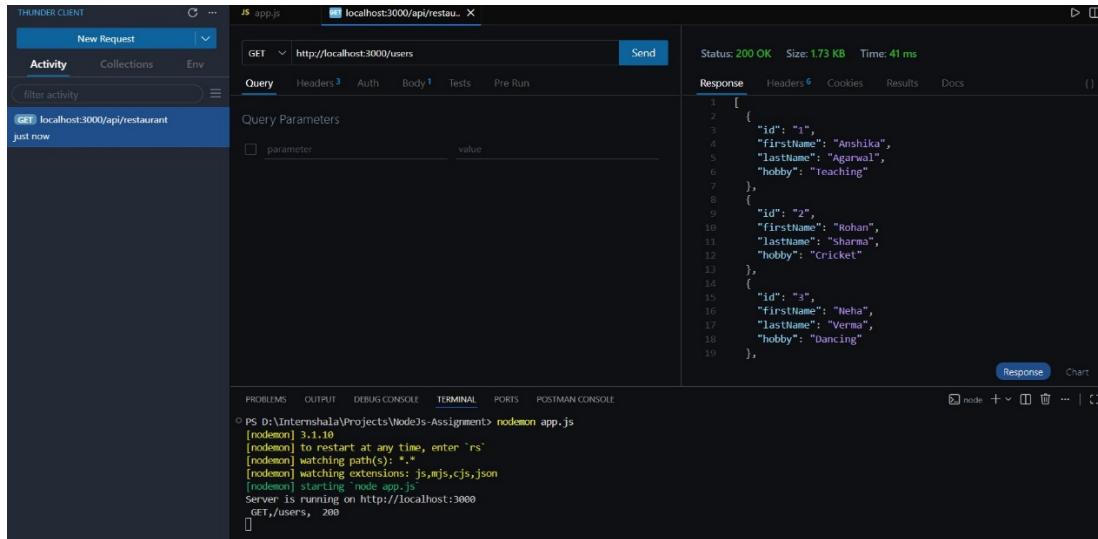
Why is this important?

It prevents storing incomplete or invalid user information. If required data is missing, a **400 (Bad Request)** response is sent with a meaningful message.

3. API Routes Explanation

GET /users — Fetch All Users

- This request retrieves the entire list of users.



The screenshot shows the Thunder Client interface. A request is being sent to `http://localhost:3000/users`. The response status is **200 OK**, size is **1.73 KB**, and time taken is **41 ms**. The response body contains an array of three user objects:

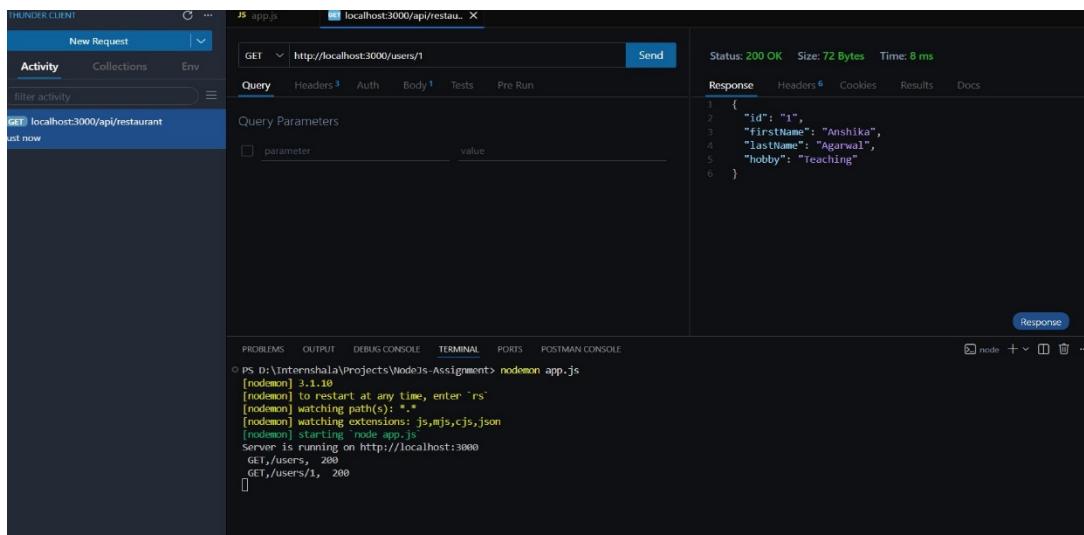
```
1 [  
2   {  
3     "id": "1",  
4     "firstName": "Anshika",  
5     "lastName": "Agarwal",  
6     "hobby": "Teaching"  
7   },  
8   {  
9     "id": "2",  
10    "firstName": "Rohan",  
11    "lastName": "Sharma",  
12    "hobby": "Cricket"  
13  },  
14  {  
15    "id": "3",  
16    "firstName": "Neha",  
17    "lastName": "Verma",  
18    "hobby": "Dancing"  
19  },
```

The terminal below shows the Node.js application starting and listening on port 3000.

- If users are available, the API returns them with a **200 (OK)** status.
- If the list doesn't contain any data, we send a message saying no users are available.

GET /users/:id — Fetch One User by ID

- This route allows us to find a single user based on their unique ID.



The screenshot shows the Thunder Client interface. A request is being sent to `http://localhost:3000/users/1`. The response status is **200 OK**, size is **72 Bytes**, and time taken is **8 ms**. The response body contains a single user object:

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

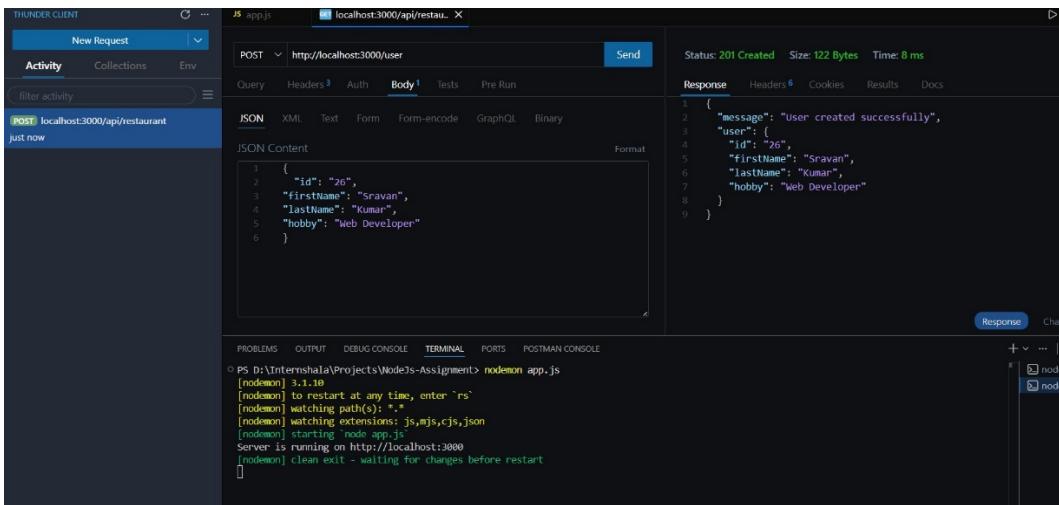
The terminal below shows the Node.js application starting and listening on port 3000.

- If a matching user is found, their information is sent back with a **200 (OK)** status.
- If the user does not exist, the API returns **404 (Not Found)** with a clear message.

POST /user — Add a New User

- This route is used to create or register a new user.

- Validation middleware ensures all required fields are provided.



THUNDER CLIENT

New Request

Activity Collections Env

POST localhost:3000/api/restaurant just now

POST http://localhost:3000/user

Send

Query Headers 3 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```

1  {
2    "id": "26",
3    "firstName": "Sravan",
4    "lastName": "Kumar",
5    "hobby": "Web Developer"
6  }

```

Status: 201 Created Size: 122 Bytes Time: 8 ms

Response Headers Cookies Results Docs

```

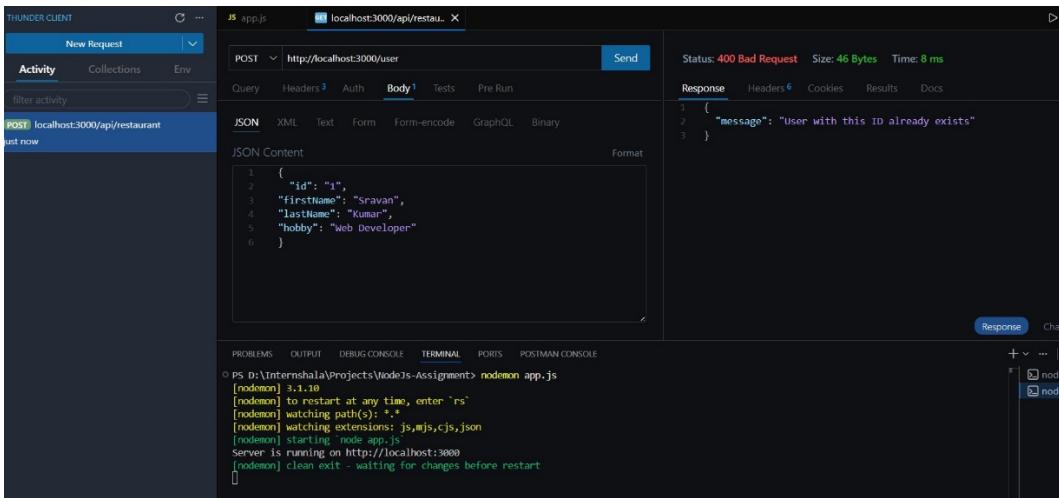
1  {
2    "message": "User created successfully",
3    "user": {
4      "id": "26",
5      "firstName": "Sravan",
6      "lastName": "Kumar",
7      "hobby": "Web Developer"
8    }
9  }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS D:\Internshala\Projects\Node.js-Assignment> nodemon app.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server is running on http://localhost:3000
[nodemon] clean exit - waiting for changes before restart

- If a user with the same ID already exists, we return a **400 (Bad Request)** message.



THUNDER CLIENT

New Request

Activity Collections Env

POST localhost:3000/api/restaurant just now

POST http://localhost:3000/user

Send

Query Headers 3 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```

1  {
2    "id": "1",
3    "firstName": "Sravan",
4    "lastName": "Kumar",
5    "hobby": "Web Developer"
6  }

```

Status: 400 Bad Request Size: 46 Bytes Time: 8 ms

Response Headers Cookies Results Docs

```

1  {
2    "message": "User with this ID already exists"
3  }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS D:\Internshala\Projects\Node.js-Assignment> nodemon app.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server is running on http://localhost:3000
[nodemon] clean exit - waiting for changes before restart

- If added successfully, a confirmation message with user details is returned with **201 (Created)**.

PUT /user/:id — Update Existing User

- This route updates the information of an existing user using their ID.

PUT http://localhost:3000/user/50

```
{
  "id": "50",
  "firstName": "ra",
  "lastName": "kumar",
  "hobby": "App Developer"
}
```

Status: 404 Not Found Size: 28 Bytes Time: 4 ms

Response Headers: Content-Type: application/json

```
{
  "message": "User not found"
}
```

- If the user does not exist, the API sends a **404 (Not Found)** message.

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

```
{
  "id": "1",
  "firstName": "Sravan",
  "lastName": "Kumar",
  "hobby": "Web Developer"
}
```

Status: 200 OK Size: 121 Bytes Time: 6 ms

Response Headers: Content-Type: application/json

```
{
  "message": "User updated successfully",
  "user": {
    "id": "1",
    "firstName": "Sravan",
    "lastName": "Kumar",
    "hobby": "Web Developer"
  }
}
```

- If the information is valid and user exists, it updates the data and returns **200 (OK)** with the updated result.
- Middleware ensures no empty data is passed while updating.

DELETE /user/:id — Remove a User

- This request removes a user.
- If the user ID doesn't exist, the API returns **404 (Not Found)**.

```

Status: 404 Not Found | Size: 28 Bytes | Time: 5 ms
Response Headers Cookies Results Docs
1 {
2   "message": "User not found"
3 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

PS D:\Internshala\Projects\Nodejs-Assignment> nodemon app.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server is running on http://localhost:3000
[nodemon] clean exit... waiting for changes before restart
PS D:\Internshala\Projects\Nodejs-Assignment> node app.js
Server is running on http://localhost:3000
PS D:\Internshala\Projects\Nodejs-Assignment>

```

- If the user exists, they are deleted and a **200 (OK)** confirmation message is returned.

```

Status: 200 OK | Size: 1.73 KB | Time: 5 ms
Response Headers Cookies Results Docs
1 [
2   {
3     "id": "2",
4     "firstName": "Rohan",
5     "lastName": "Sharma",
6     "hobby": "Cricket"
7   },
8   {
9     "id": "3",
10    "firstName": "Neha",
11    "lastName": "Verma",
12    "hobby": "Dancing"
13  },
14  {
15    "id": "4",
16    "firstName": "Aman",
17    "lastName": "Kumar",
18    "hobby": "Football"
19  }
20 ]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

PS D:\Internshala\Projects\Nodejs-Assignment> nodemon app.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server is running on http://localhost:3000
[nodemon] clean exit... waiting for changes before restart
PS D:\Internshala\Projects\Nodejs-Assignment> node app.js
Server is running on http://localhost:3000
PS D:\Internshala\Projects\Nodejs-Assignment>

```

We can find user after deletion.

```

Status: 200 OK | Size: 1.73 KB | Time: 5 ms
Response Headers Cookies Results Docs
1 [
2   {
3     "id": "2",
4     "firstName": "Rohan",
5     "lastName": "Sharma",
6     "hobby": "Cricket"
7   },
8   {
9     "id": "3",
10    "firstName": "Neha",
11    "lastName": "Verma",
12    "hobby": "Dancing"
13  },
14  {
15    "id": "4",
16    "firstName": "Aman",
17    "lastName": "Kumar",
18    "hobby": "Football"
19  }
20 ]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

PS D:\Internshala\Projects\Nodejs-Assignment> nodemon app.js
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server is running on http://localhost:3000
[nodemon] clean exit... waiting for changes before restart
PS D:\Internshala\Projects\Nodejs-Assignment> node app.js
Server is running on http://localhost:3000
PS D:\Internshala\Projects\Nodejs-Assignment>

```

4. Error Handling Summary

A global error handler makes sure:

- The server doesn't crash if anything goes wrong internally.
- All errors return understandable messages instead of raw system errors.

- Proper status codes such as **500 (Server Error)** or specific ones like **404**, **400**, **403**, etc., are sent.