



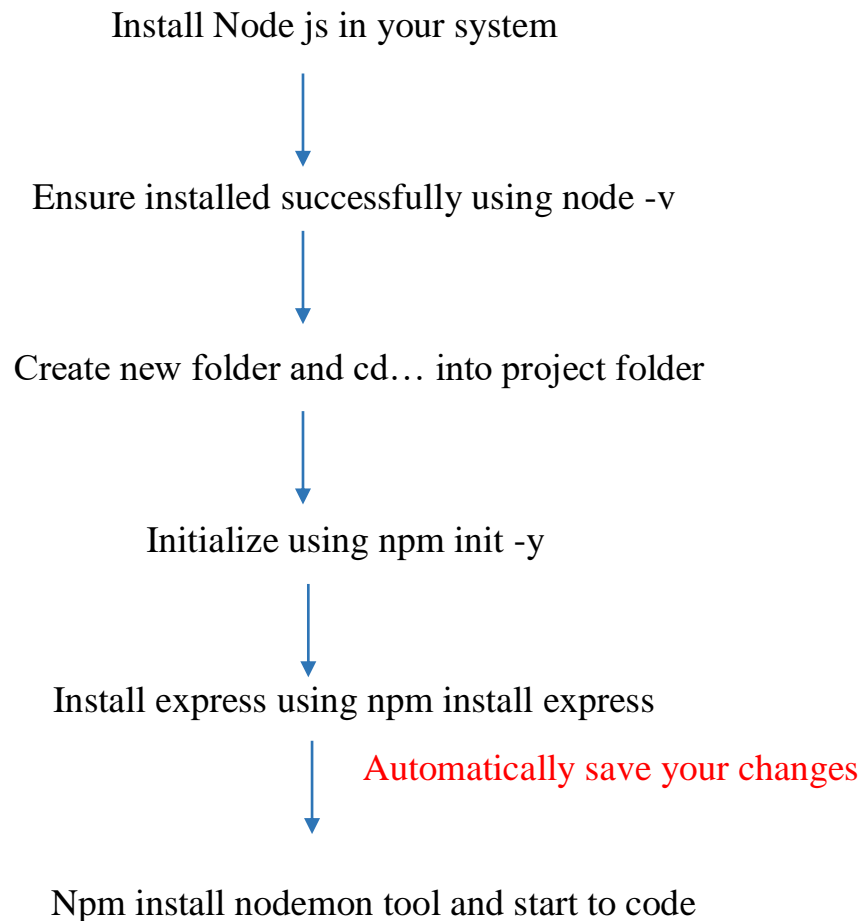
ASSIGNMENT-1

BUILD A RESTFUL API USING **NODE.JS AND EXPRESS**

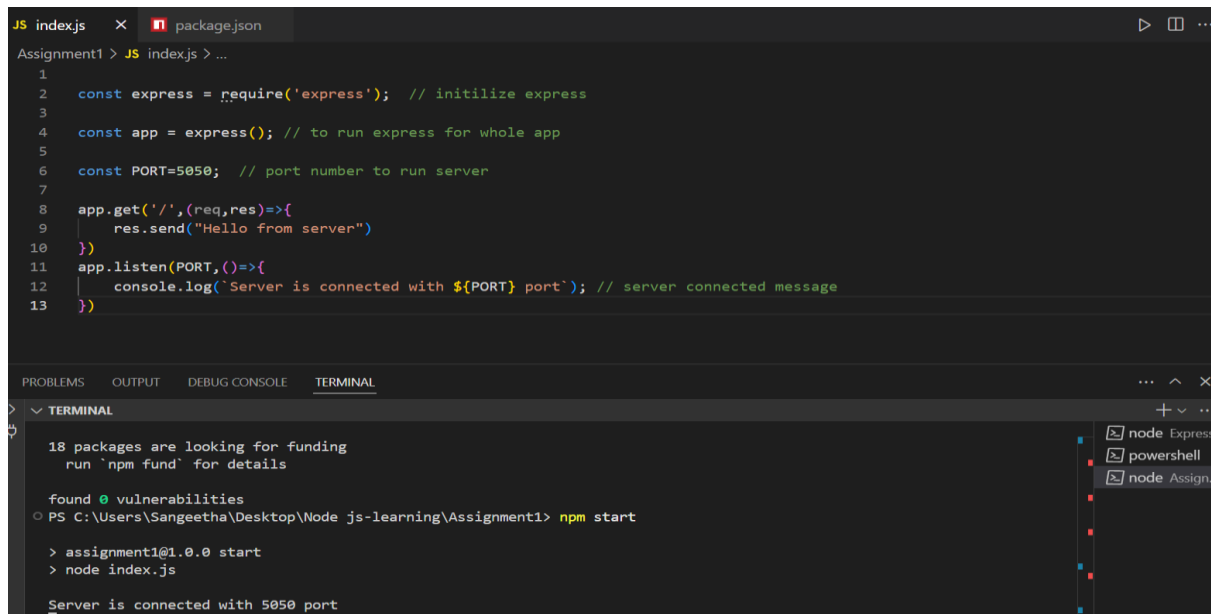
Objective of the task:

Create a simple RESTful API for managing a list of users, testing concepts such as routing, middleware, HTTP methods, status codes, error handling, and interaction with a data source.

Initialize Node js project:



Set-Up Node js Project



```
JS index.js  X  package.json
Assignment1 > JS index.js > ...
1
2  const express = require('express'); // initialize express
3
4  const app = express(); // to run express for whole app
5
6  const PORT=5050; // port number to run server
7
8  app.get('/',(req,res)=>{
9    res.send("Hello from server")
10 })
11 app.listen(PORT,()=>{
12   console.log(`Server is connected with ${PORT} port`); // server connected message
13 })
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

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

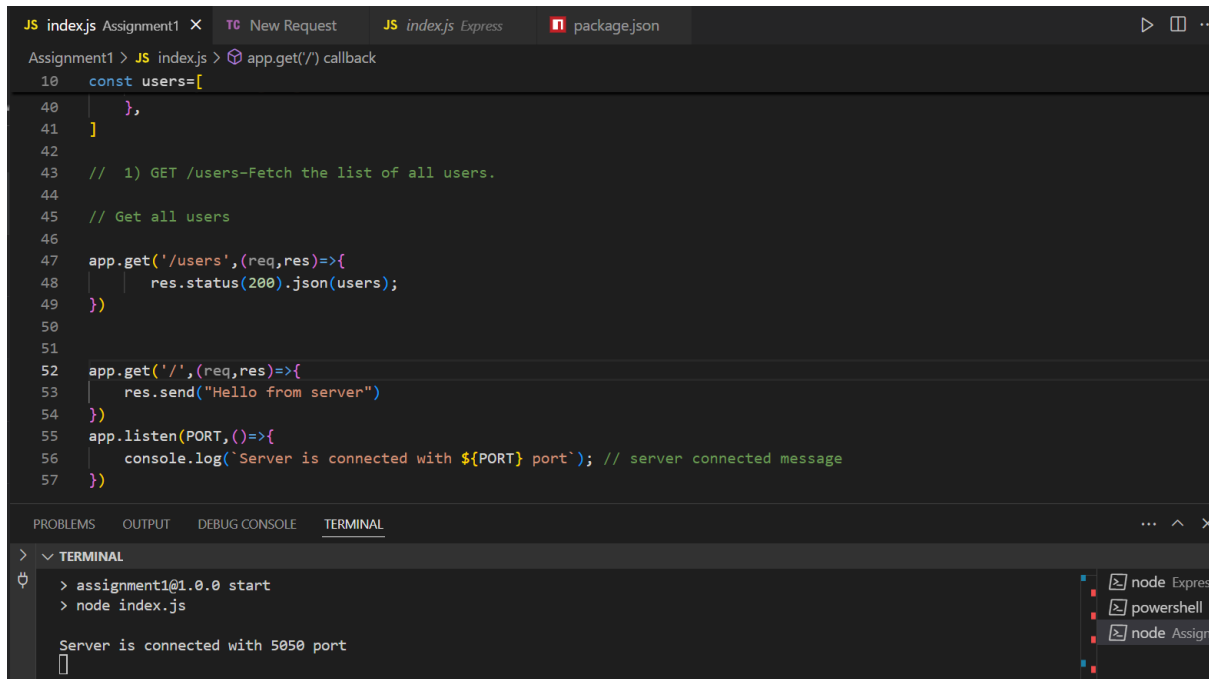
found 0 vulnerabilities
PS C:\Users\Sangeetha\Desktop\Node js-learning\Assignment1> npm start

> assignment1@1.0.0 start
> node index.js

Server is connected with 5050 port
```

Task 1

GET /users Fetch the list of all users



The screenshot shows the VS Code editor with the `index.js` file open. The code defines a list of users and a GET endpoint `/users` that returns them as JSON. The terminal shows the server starting on port 5050.

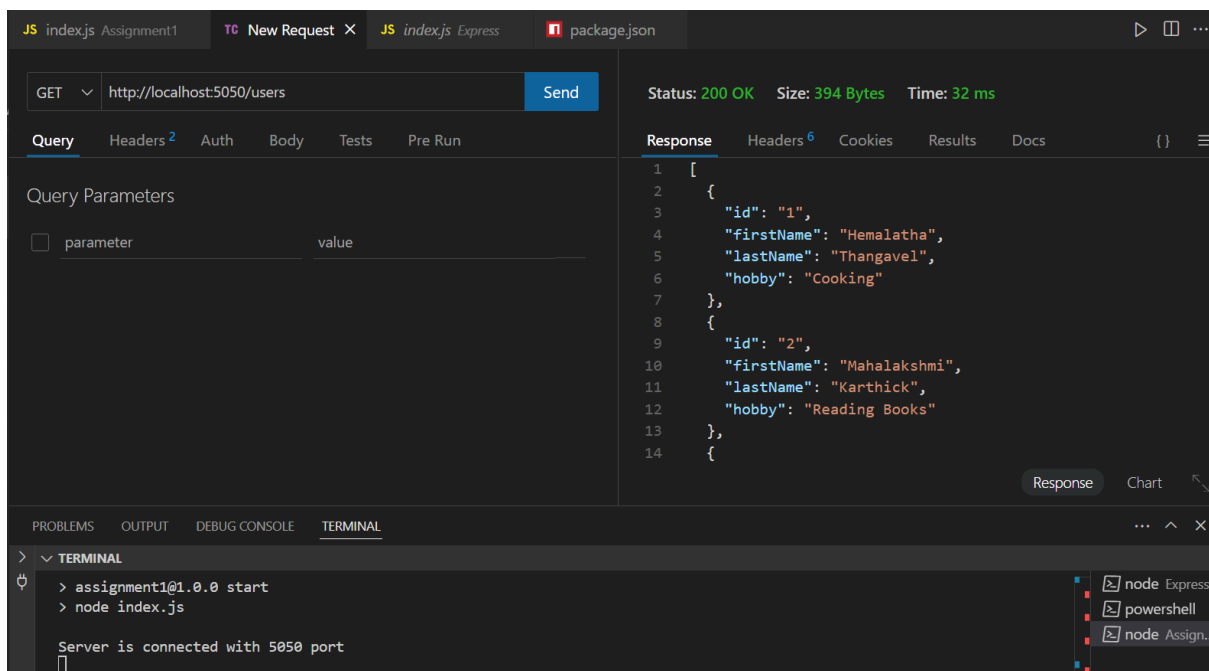
```
10 const users=[
40   },
41 ]
42
43 // 1) GET /users-Fetch the list of all users.
44
45 // Get all users
46
47 app.get('/users',(req,res)=>{
48   res.status(200).json(users);
49 })
50
51
52 app.get('/',(req,res)=>{
53   res.send("Hello from server")
54 })
55 app.listen(PORT,()=>{
56   console.log(`Server is connected with ${PORT} port`); // server connected message
57 })
```

Terminal Output:

```
> assignment1@1.0.0 start
> node index.js

Server is connected with 5050 port
```

Test code using Thunder- Client



The screenshot shows the Thunder Client interface. A GET request to `http://localhost:5050/users` was sent successfully, returning a 200 OK status. The response body contains a JSON array of two users.

Request Details:

- Method: GET
- URL: `http://localhost:5050/users`

Response Details:

- Status: 200 OK
- Size: 394 Bytes
- Time: 32 ms

Response Body:

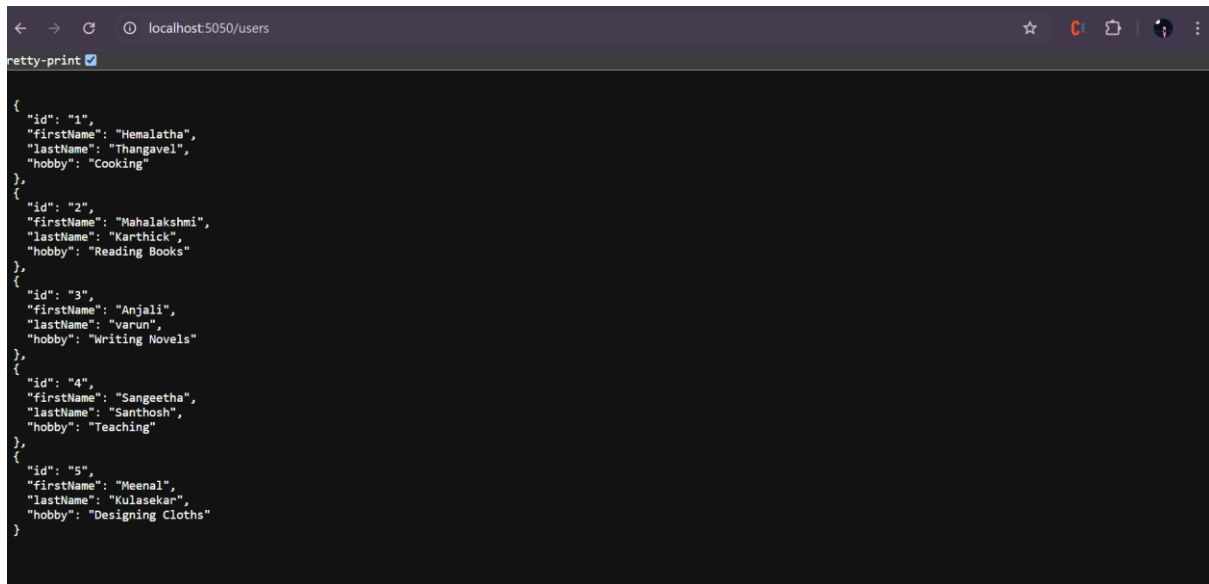
```
1 [
2   {
3     "id": "1",
4     "firstName": "Hemalatha",
5     "lastName": "Thangavel",
6     "hobby": "Cooking"
7   },
8   {
9     "id": "2",
10    "firstName": "Mahalakshmi",
11    "lastName": "Karthick",
12    "hobby": "Reading Books"
13  },
14 ]
```

Terminal Output:

```
> assignment1@1.0.0 start
> node index.js

Server is connected with 5050 port
```

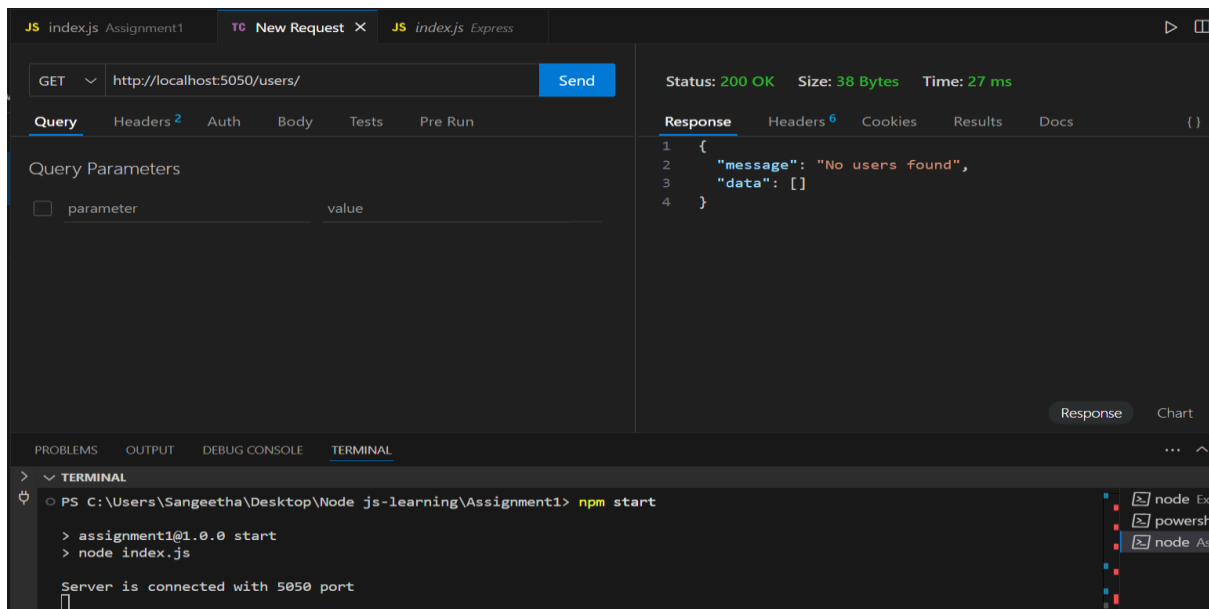
API view in browser



A screenshot of a web browser window displaying a JSON array of user data. The address bar shows the URL `localhost:5050/users`. The page content is a JSON array with five user objects, each containing `id`, `firstName`, `lastName`, and `hobby` fields. A 'pretty-print' checkbox is visible in the top left corner of the browser window.

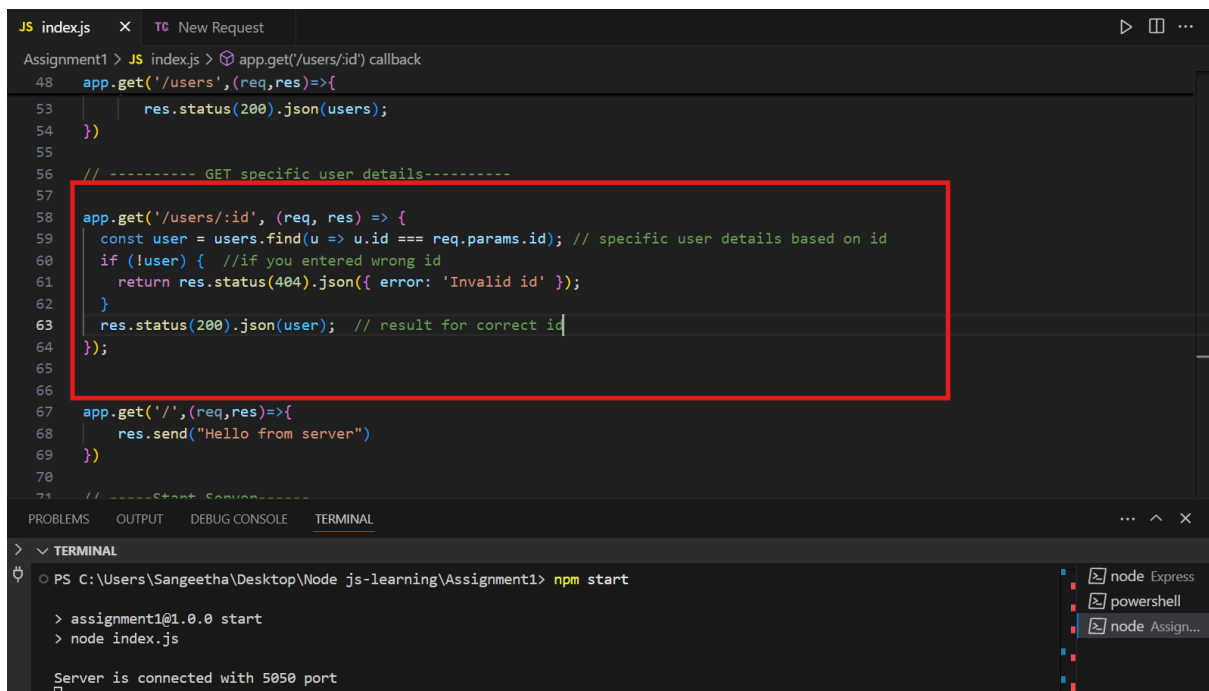
```
{
  "id": "1",
  "firstName": "Hemalatha",
  "lastName": "Thangavel",
  "hobby": "cooking"
},
{
  "id": "2",
  "firstName": "Mahalakshmi",
  "lastName": "Karthick",
  "hobby": "Reading Books"
},
{
  "id": "3",
  "firstName": "Anjali",
  "lastName": "varun",
  "hobby": "Writing Novels"
},
{
  "id": "4",
  "firstName": "Sangeetha",
  "lastName": "Santhosh",
  "hobby": "Teaching"
},
{
  "id": "5",
  "firstName": "Meenal",
  "lastName": "Kulasekar",
  "hobby": "Designing Cloths"
}
}
```

Error : If the data array is empty – display error message



Task – 2

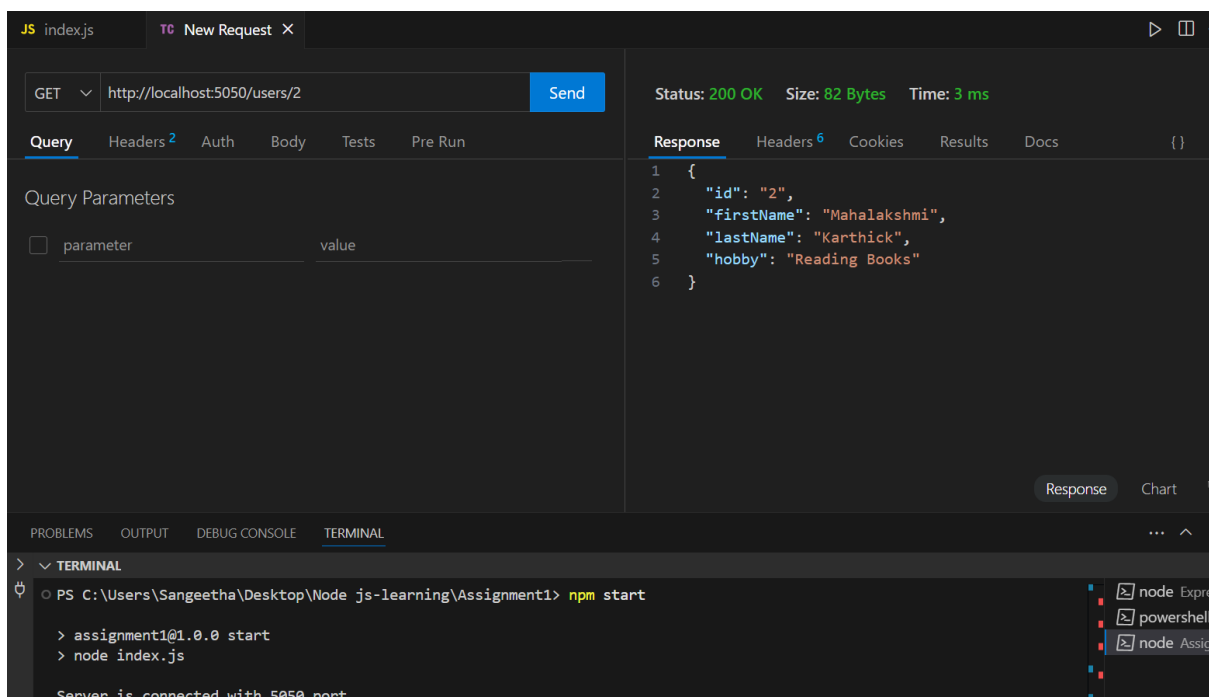
GET /users/:id–Fetch details of a specific user by ID



The screenshot shows a VS Code editor with a file named `index.js`. The code implements an Express.js application. A red box highlights the `app.get('/users/:id')` route handler. Below the code, the terminal shows the server starting successfully on port 5050.

```
JS index.js x TC New Request
Assignment1 > JS index.js > app.get('/users/:id') callback
48 app.get('/users/',(req,res)=>{
53   res.status(200).json(users);
54 })
55
56 // ----- GET specific user details-----
57
58 app.get('/users/:id', (req, res) => {
59   const user = users.find(u => u.id === req.params.id); // specific user details based on id
60   if (!user) { //if you entered wrong id
61     return res.status(404).json({ error: 'Invalid id' });
62   }
63   res.status(200).json(user); // result for correct id
64 });
65
66 app.get('/',(req,res)=>{
67   res.send("Hello from server")
68 })
69
70
71 // -----Start Server-----
72
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> v TERMINAL
PS C:\Users\Sangeetha\Desktop\Node js-learning\Assignment1> npm start
> assignment1@1.0.0 start
> node index.js
Server is connected with 5050 port
```

Result in Thunder-client



The screenshot shows the Thunder Client interface. A GET request to `http://localhost:5050/users/2` has been sent, resulting in a 200 OK status. The response body is a JSON object containing user details. The terminal at the bottom shows the server is running.

GET http://localhost:5050/users/2 Send

Status: 200 OK Size: 82 Bytes Time: 3 ms

Query Parameters

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

Response

```
1 {
2   "id": "2",
3   "firstName": "Mahalakshmi",
4   "lastName": "Karthick",
5   "hobby": "Reading Books"
6 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Sangeetha\Desktop\Node js-learning\Assignment1> npm start
> assignment1@1.0.0 start
> node index.js
Server is connected with 5050 port
```

Error message – If you entered invalid user id

The screenshot displays the VS Code interface with a REST client tab and a terminal window.

REST Client Tab:

- Method:** GET
- URL:** `http://localhost:5050/users/23`
- Status:** 404 Not Found
- Size:** 22 Bytes
- Time:** 6 ms
- Response:**

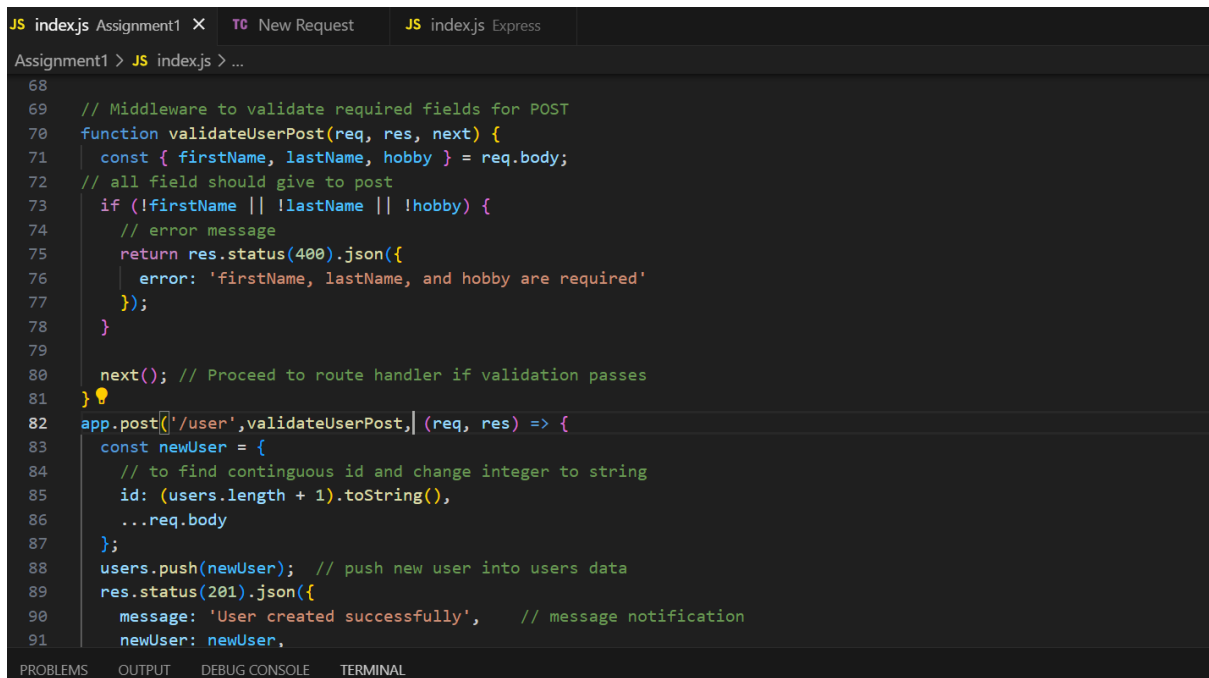
```
1 {  
2   "error": "Invalid id"  
3 }
```

Terminal Window:

```
PS C:\Users\Sangeetha\Desktop\Node js-learning\Assignment1> npm start  
  
> assignment1@1.0.0 start  
> node index.js  
  
Server is connected with 5050 port
```

Task – 3

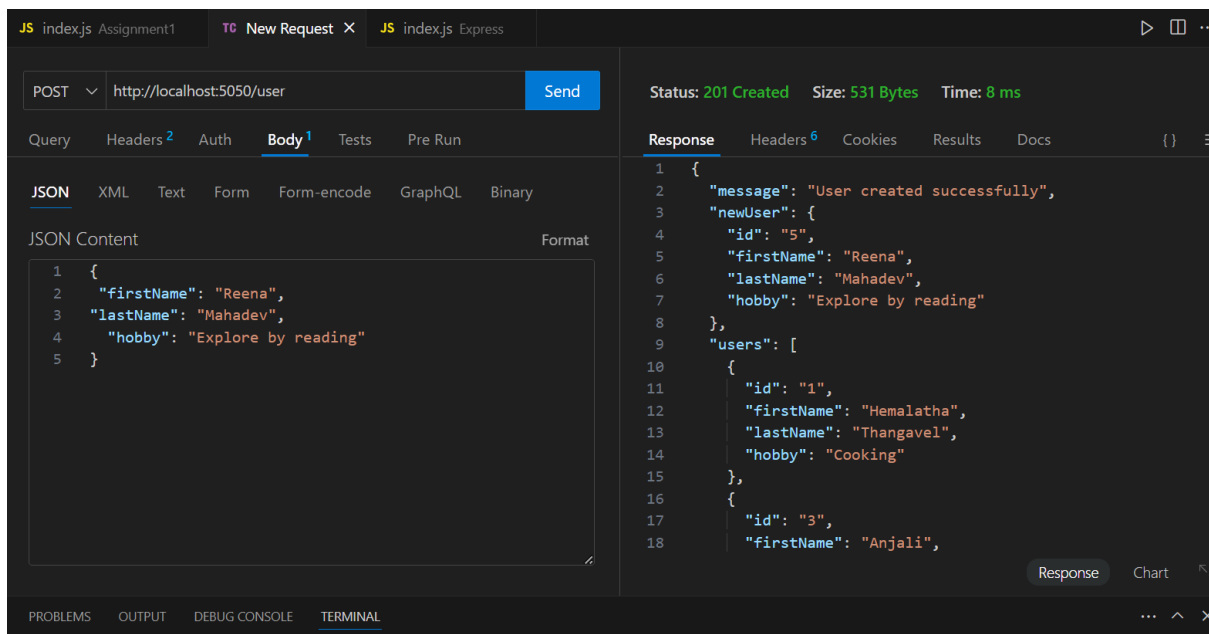
POST /user–Add a new user



```
JS index.js Assignment1 X TC New Request JS index.js Express
Assignment1 > JS index.js > ...
68
69 // Middleware to validate required fields for POST
70 function validateUserPost(req, res, next) {
71   const { firstName, lastName, hobby } = req.body;
72   // all field should give to post
73   if (!firstName || !lastName || !hobby) {
74     // error message
75     return res.status(400).json({
76       error: 'firstName, lastName, and hobby are required'
77     });
78   }
79
80   next(); // Proceed to route handler if validation passes
81 }
82 app.post('/user', validateUserPost, (req, res) => {
83   const newUser = {
84     // to find contiguous id and change integer to string
85     id: (users.length + 1).toString(),
86     ...req.body
87   };
88   users.push(newUser); // push new user into users data
89   res.status(201).json({
90     message: 'User created successfully', // message notification
91     newUser: newUser,
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Response in Thunder-client



POST http://localhost:5050/user Send

Status: 201 Created Size: 531 Bytes Time: 8 ms

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "firstName": "Reena",
3   "lastName": "Mahadev",
4   "hobby": "Explore by reading"
5 }
```

Response Headers Cookies Results Docs {}

```
1 {
2   "message": "User created successfully",
3   "newUser": {
4     "id": "5",
5     "firstName": "Reena",
6     "lastName": "Mahadev",
7     "hobby": "Explore by reading"
8   },
9   "users": [
10    {
11      "id": "1",
12      "firstName": "Hemalatha",
13      "lastName": "Thangavel",
14      "hobby": "Cooking"
15    },
16    {
17      "id": "3",
18      "firstName": "Anjali",
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Result in Browser

```
Pretty-print ☒
{
  "id": "1",
  "firstName": "Hemalatha",
  "lastName": "Thangavel",
  "hobby": "Cooking"
},
{
  "id": "2",
  "firstName": "Mahalakshmi",
  "lastName": "Karthick",
  "hobby": "Reading Books"
},
{
  "id": "3",
  "firstName": "Anjali",
  "lastName": "varun",
  "hobby": "Writing Novels"
},
{
  "id": "4",
  "firstName": "Sangeetha",
  "lastName": "Santhosh",
  "hobby": "Teaching"
},
{
  "id": "5",
  "firstName": "Meenal",
  "lastName": "Kulasekar",
  "hobby": "Designing Cloths"
},
{
  "id": "6",
  "firstName": "Maya",
  "lastName": "Arjun",
  "hobby": "Drawing natural scenarios"
}
}
```


Task – 4

PUT /user/:id-Update details of an existing user

```
JS indexjs Assignment1 X TC New Request JS indexjs Express
Assignment1 > JS indexjs > ...
83 //----- PUT method for updation-----
84
85 function validateUserPut(req, res, next) {
86   const { firstName, lastName, hobby } = req.body;
87   if (!firstName && !lastName && !hobby) {
88     return res.status(400).json({ error: 'At least one field (firstName, lastName, or hobby) is required to update' });
89   }
90   next();
91 }
92
93 app.put('/user/:id', validateUserPut, (req, res) => {
94   // finding id
95   const userIndex = users.findIndex(u => u.id === req.params.id);
96   // error message 404
97   if (userIndex === -1) {
98     return res.status(404).json({ error: 'User not found' });
99   }
100   // Merge existing user with new fields
101   users[userIndex] = {
102     ...users[userIndex], // existing user data
103     ...req.body           // overwrite with provided fields
104   };
105
106   // updated message
107   res.status(200).json({
108     message: 'User updated successfully',
109     updatedUser: users[userIndex],
110     users: users
111   });
112 }
```

Response in Thunder-client

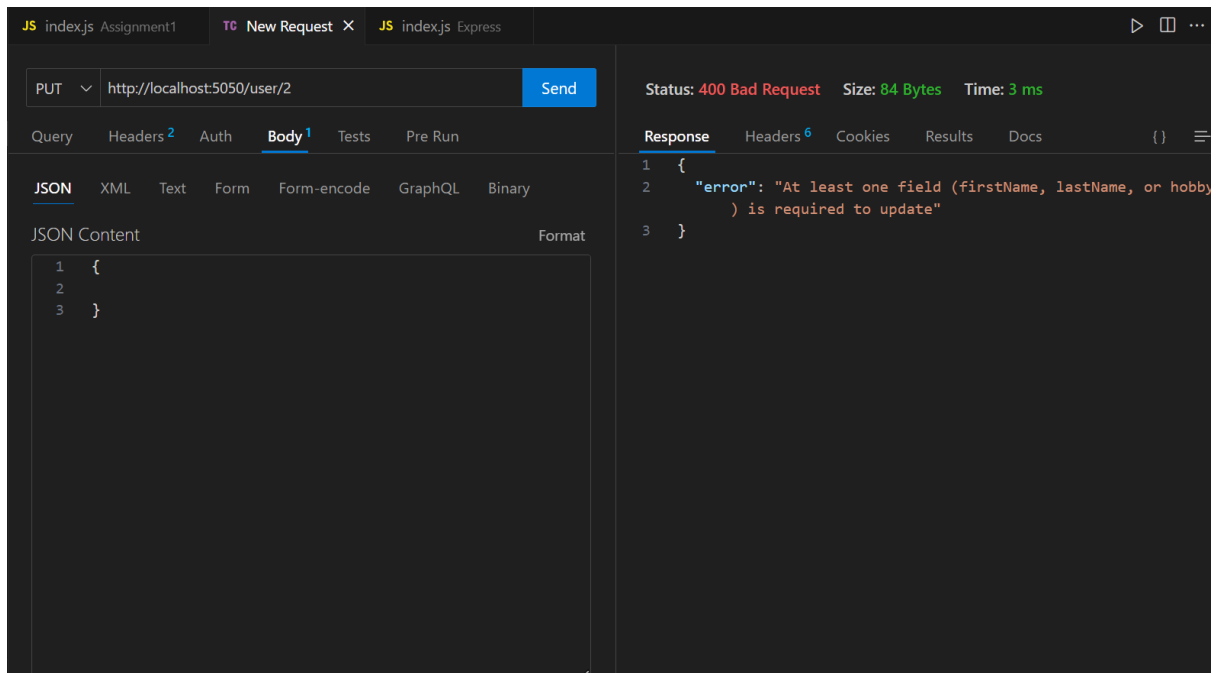
The screenshot shows the Thunder-client interface with a PUT request to `http://localhost:5050/user/2` sent successfully. The response status is 200 OK, with a size of 529 Bytes and a time of 12 ms. The response body is a JSON array of three user objects. The first user object is the one that was updated, showing the new first name 'Hemalatha' and hobby 'Cooking'. The other two user objects remain unchanged.

```
PUT http://localhost:5050/user/2
Status: 200 OK Size: 529 Bytes Time: 12 ms

Response
8 },
9 "users": [
10   {
11     "id": "1",
12     "firstName": "Hemalatha",
13     "lastName": "Thangavel",
14     "hobby": "Cooking"
15   },
16   {
17     "id": "2",
18     "firstName": "Prabha",
19     "lastName": "Karthick",
20     "hobby": "playing games"
21   },
22   {
23     "id": "3",
24     "firstName": "Anjali",
25     "lastName": "Varun",
26     "hobby": "Writing Novels"
27   }
28 ]
```

Error message

If no field is entered and try to make PUT request



Task – 5

DELETE /user/:id–Delete a user by ID

```
JS index.js Assignment1 X TC New Request JS index.js Express
Assignment1 > JS index.js > app.delete('/user/:id') callback
115
116
117 // -----DELETE request -----
118
119 app.delete('/user/:id', (req, res) => {
120   const userIndex = users.findIndex(u => u.id === req.params.id); // id checking
121
122   if (userIndex === -1) { // error message
123     return res.status(404).json({ error: 'User not found' });
124   }
125
126   const deletedUser = users.splice(userIndex, 1); // deletes user at that index
127
128   // display if successfully deleted the user details
129   res.status(200).json({
130     message: 'User deleted',
131     user: deletedUser[0],
132     // shows remaining users data
133     remainingUsers: users
134   });
135 });
136
137
138
```

Response in Thunder-client

JS index.js Assignment1 TC New Request * X JS index.js Express

DELETE http://localhost:5050/user/2 Send

Status: 200 OK Size: 445 Bytes Time: 56 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

Form Fields ☐ Files Import

field name value

Learn more about how to set a custom [content-type](#) for a field.

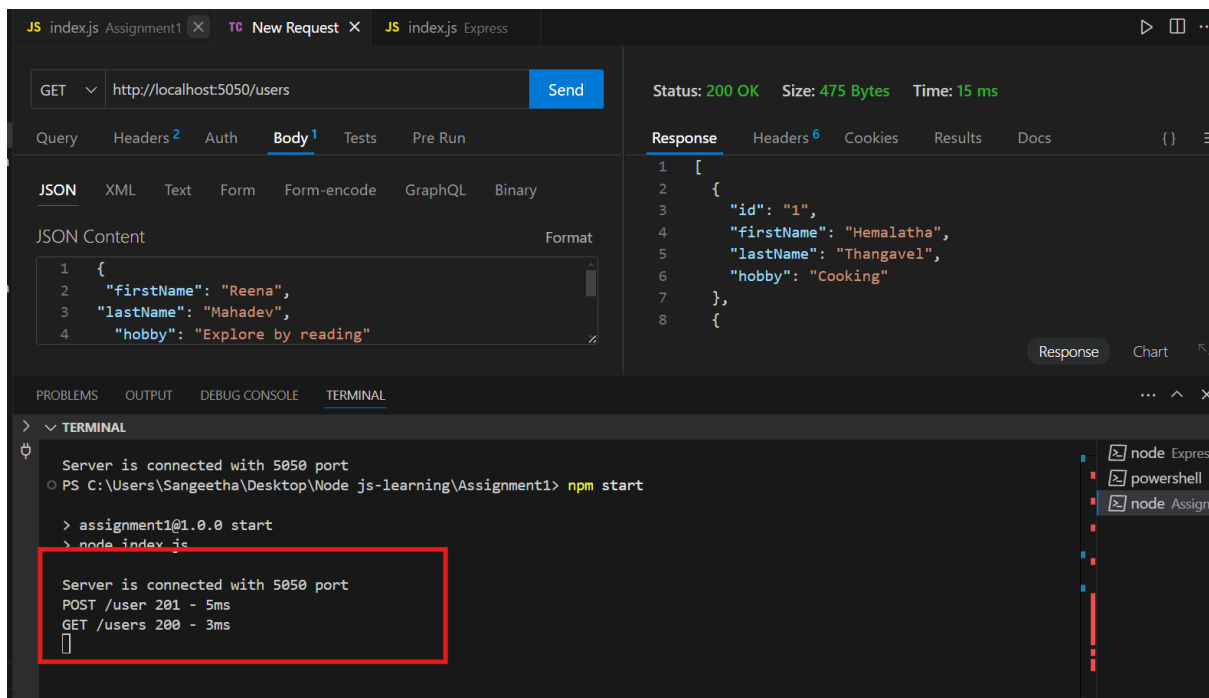
Response Headers 6 Cookies Results Docs {}

```
1 {
2   "message": "User deleted",
3   "user": {
4     "id": "2",
5     "firstName": "Mahalakshmi",
6     "lastName": "Karthick",
7     "hobby": "Reading Books"
8   },
9   "remainingUsers": [
10    {
11      "id": "1",
12      "firstName": "Hemalatha",
13      "lastName": "Thangavel",
14      "hobby": "Cooking"
15    },
16    {
17      "id": "3",
18      "firstName": "Anjali",
```

Response Chart

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Use middleware to log the details of each request



Conclusion

This RESTful API allows creating, reading, updating, and deleting user data using HTTP methods.

- GET -- retrieves all users or a specific user by ID.
- POST-- adds a new user with validation.
- PUT -- updates user details partially or fully.
- DELETE-- removes a user by ID and returns the updated list.

GitHub Link

<https://github.com/Hema2802/create-RESTfull-API>