

Batch:D-2

Roll No.:16010123325

Experiment / assignment / tutorial No._7

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Create a RESTful API server in Express and Node.js. Implementation + Testing application using postman/Thurderclient

AIM: Create a RESTful API server in Express and Node.js. Implementation + Testing application using postman/Thurderclient

Problem Definition: To develop and test a RESTful API using Node.js and Express.js that performs CRUD (Create, Read, Update, Delete) operations on student data. The API should handle JSON requests and responses, and be tested using Thunder Client to verify proper backend functionality.

Resources used:

- Software:
 - Node.js
 - Visual Studio Code
 - Thunder Client (VS Code Extension)
- Libraries:
 - Express.js
 - Body-Parser (for JSON parsing)

Expected OUTCOME of Experiment:

Successful implementation of a RESTful API that performs CRUD operations using Express.js and Node.js, with verification through Thunder Client showing correct HTTP responses and JSON data for each operation.

CO 3: Test the concepts and components of various front-end, back-end web app development technologies & frameworks using web development tools.

Books/ Journals/ Websites referred:

1. Official Express.js Documentation – <https://expressjs.com>
2. MDN Web Docs – REST API Basics and HTTP Methods

Pre Lab/ Prior Concepts:

Thunder Client is a lightweight REST API client available as an extension inside VS Code. It allows developers to test their backend APIs by sending HTTP requests (GET, POST, PUT, DELETE) and receiving responses directly in JSON format. It simplifies testing without needing an external tool like Postman.

Methodology:

1. Initialize a Node.js project using npm init -y.
2. Install Express.js using npm install express.
3. Create a server.js file to define CRUD (Create, Read, Update, Delete) routes.
4. Run the server using node server.js.
5. Use Thunder Client to test each API endpoint.

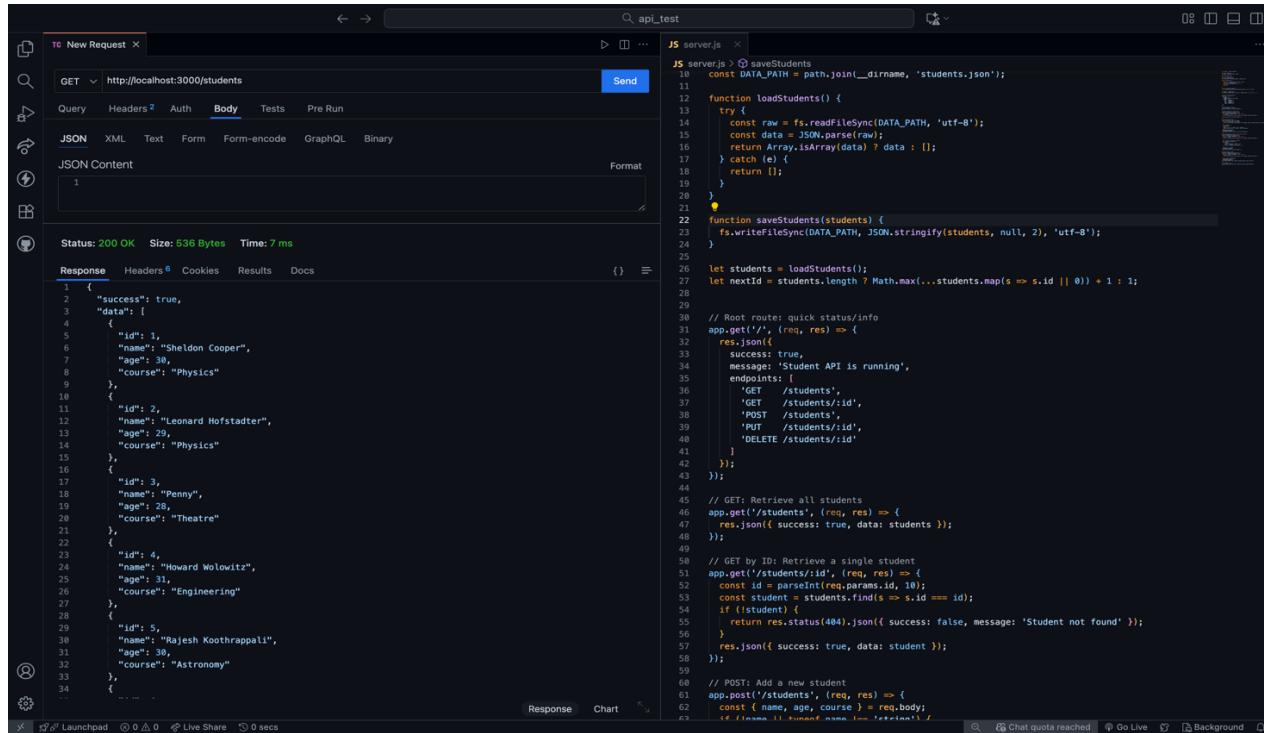
Implementation Details:

The implementation includes defining RESTful endpoints using Express.js. Each endpoint corresponds to a CRUD operation on a simple student dataset.

Steps for execution:

1. Start the server using the command: node server.js
2. Open Thunder Client from the VS Code sidebar.
3. Perform the following operations:
 - GET: Retrieve all students.
 - POST: Add a new student.
 - GET by ID: Retrieve a single student.
 - PUT: Update student details.
 - DELETE: Remove a student.

GET Request – Fetch All Students

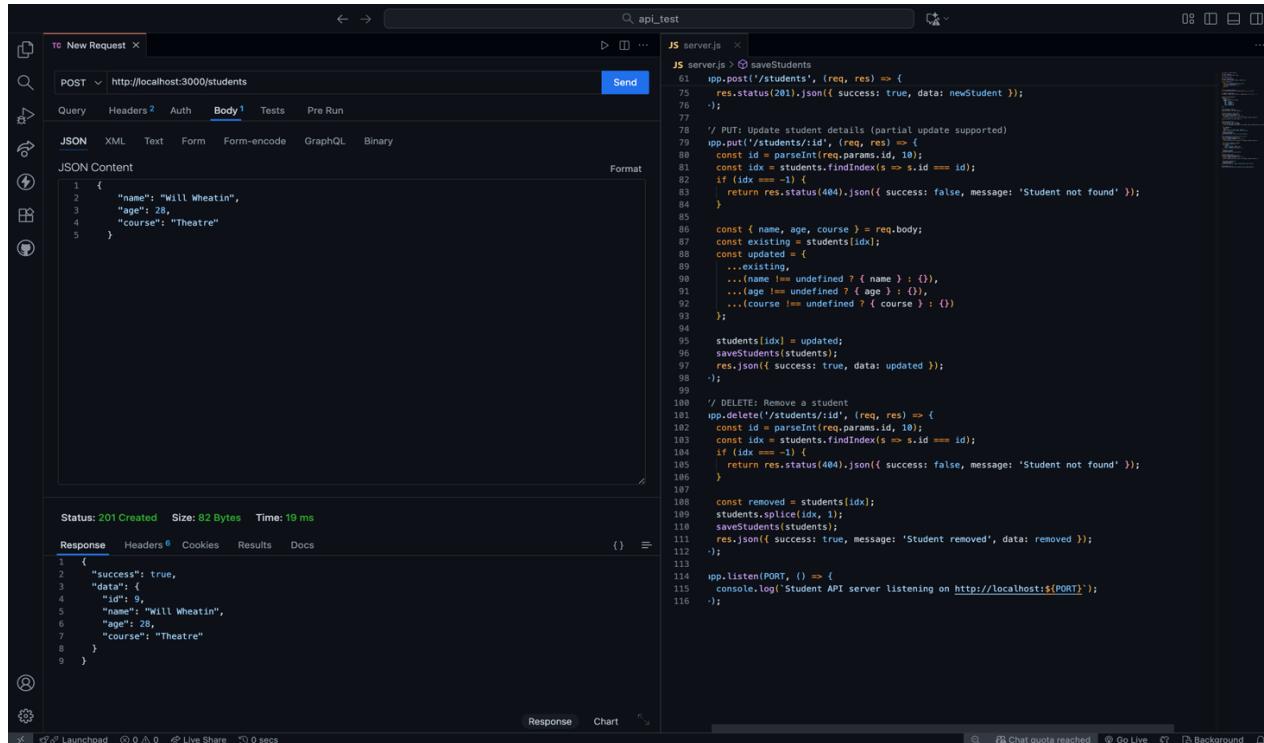


```

1 {
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "Sheldon Cooper",
      "age": 38,
      "course": "Physics"
    },
    {
      "id": 2,
      "name": "Leonard Hofstadter",
      "age": 29,
      "course": "Physics"
    },
    {
      "id": 3,
      "name": "Penny",
      "age": 28,
      "course": "Theatre"
    },
    {
      "id": 4,
      "name": "Howard Wolowitz",
      "age": 31,
      "course": "Engineering"
    },
    {
      "id": 5,
      "name": "Rajesh Koothrappali",
      "age": 38,
      "course": "Astronomy"
    }
  ]
}

```

POST Request – Add New Student

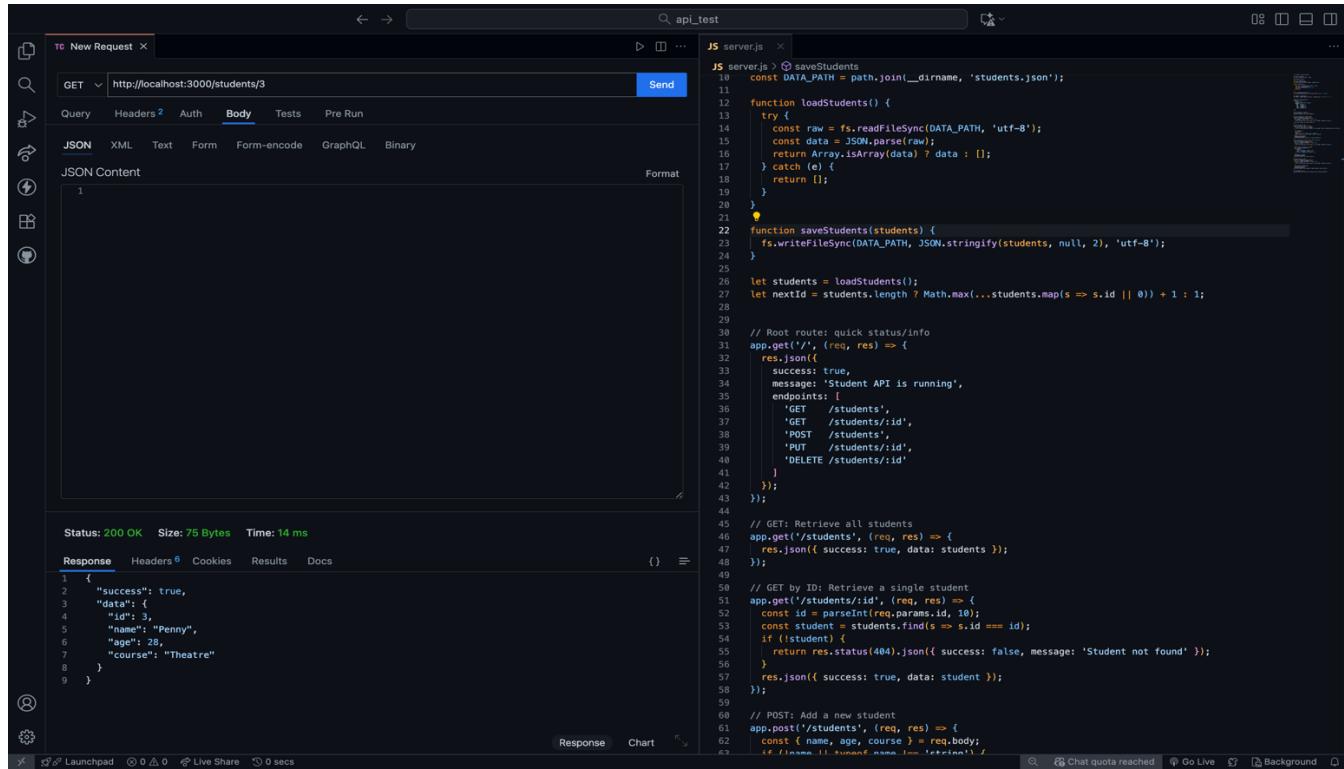


```

1 {
  "success": true,
  "data": {
    "id": 9,
    "name": "Will Wheatin",
    "age": 28,
    "course": "Theatre"
  }
}

```

GET by ID Request – Fetch Single Student



Status: 200 OK Size: 76 Bytes Time: 14 ms

```

1 {
2   "success": true,
3   "data": {
4     "id": 3,
5     "name": "Penny",
6     "age": 28,
7     "course": "Theatre"
8   }
9 }

```

Response Headers Cookies Results Docs

```

1 {
2   "success": true,
3   "data": {
4     "id": 3,
5     "name": "Penny",
6     "age": 28,
7     "course": "Theatre"
8   }
9 }

```

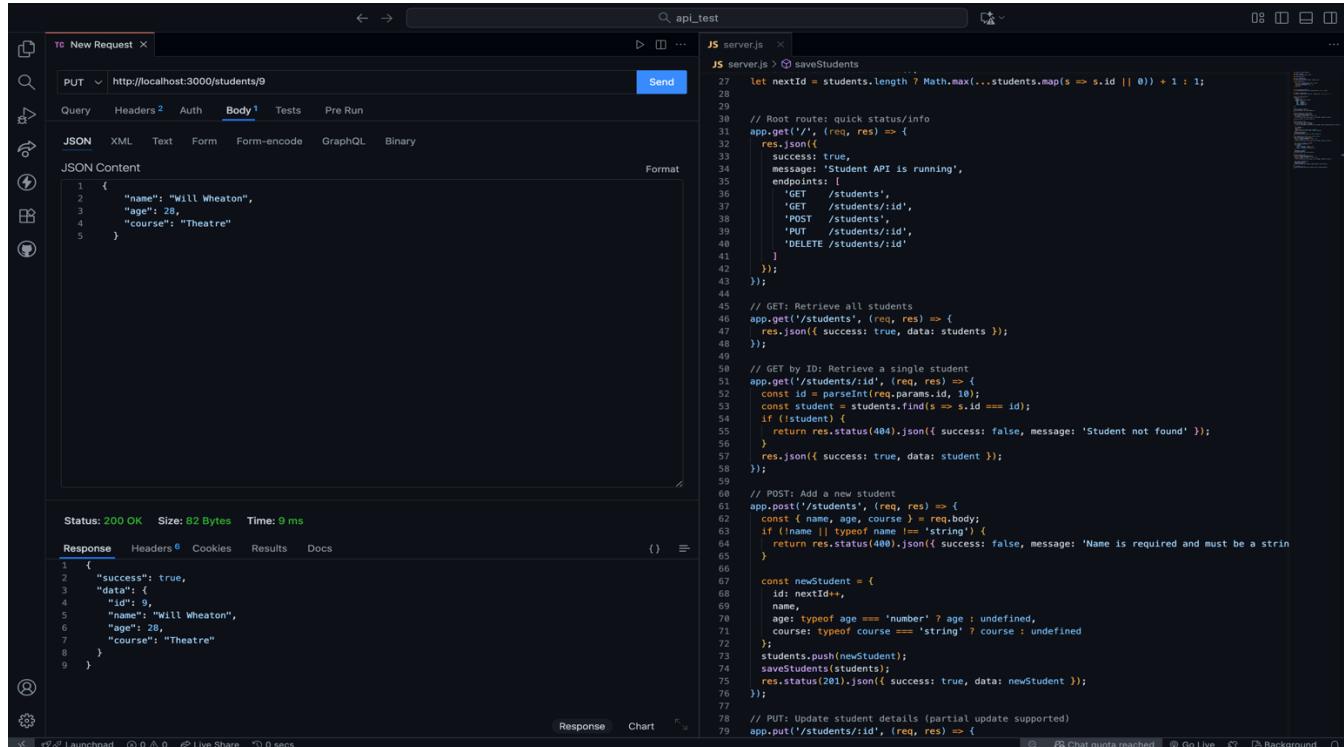
JS server.js x

```

16 const DATA_PATH = path.join(__dirname, 'students.json');
17
18 function loadStudents() {
19   try {
20     const raw = fs.readFileSync(DATA_PATH, 'utf-8');
21     const data = JSON.parse(raw);
22     return Array.isArray(data) ? data : [];
23   } catch (e) {
24     return [];
25   }
26 }
27
28 function saveStudents(students) {
29   fs.writeFileSync(DATA_PATH, JSON.stringify(students, null, 2), 'utf-8');
30 }
31
32 let students = loadStudents();
33 let nextId = students.length ? Math.max(...students.map(s => s.id || 0)) + 1 : 1;
34
35 // Root route: quick status/info
36 app.get('/', (req, res) => {
37   res.json({
38     success: true,
39     message: 'Student API is running',
40     endpoints: [
41       'GET /students',
42       'GET /students/:id',
43       'POST /students',
44       'PUT /students/:id',
45       'DELETE /students/:id'
46     ]
47   });
48 })
49
50 // GET: Retrieve all students
51 app.get('/students', (req, res) => {
52   res.json({ success: true, data: students });
53 })
54
55 // GET by ID: Retrieve a single student
56 app.get('/students/:id', (req, res) => {
57   const id = parseInt(req.params.id, 10);
58   const student = students.find(s => s.id === id);
59   if (!student) {
60     return res.status(404).json({ success: false, message: 'Student not found' });
61   }
62   res.json({ success: true, data: student });
63 })
64
65 // POST: Add a new student
66 app.post('/students', (req, res) => {
67   const { name, age, course } = req.body;
68   if (!name || typeof name !== 'string') {
69     return res.status(400).json({ success: false, message: 'Name is required and must be a string' });
70   }
71
72   const newStudent = {
73     id: nextId++,
74     name,
75     age: typeof age === 'number' ? age : undefined,
76     course: typeof course === 'string' ? course : undefined
77   };
78   students.push(newStudent);
79   saveStudents(students);
80   res.status(201).json({ success: true, data: newStudent });
81 })
82
83 // PUT: Update student details (partial update supported)
84 app.put('/students/:id', (req, res) => {
85   const id = parseInt(req.params.id, 10);
86   const student = students.find(s => s.id === id);
87   if (!student) {
88     return res.status(404).json({ success: false, message: 'Student not found' });
89   }
90
91   const updatedStudent = {
92     id: student.id,
93     name: req.body.name || student.name,
94     age: req.body.age || student.age,
95     course: req.body.course || student.course
96   };
97   students[students.indexOf(student)] = updatedStudent;
98   saveStudents(students);
99   res.status(200).json({ success: true, data: updatedStudent });
100 })
101
102 // DELETE: Remove a student
103 app.delete('/students/:id', (req, res) => {
104   const id = parseInt(req.params.id, 10);
105   const studentIndex = students.findIndex(s => s.id === id);
106   if (studentIndex === -1) {
107     return res.status(404).json({ success: false, message: 'Student not found' });
108   }
109   students.splice(studentIndex, 1);
110   saveStudents(students);
111   res.status(200).json({ success: true, message: 'Student removed' });
112 })
113
114 // Error handling
115 app.use((err, req, res, next) => {
116   console.error(err.message);
117   res.status(500).json({ success: false, message: 'Internal Server Error' });
118 })
119
120 // Listen for requests
121 app.listen(3000, () => {
122   console.log('Student API is running on port 3000');
123 })

```

PUT Request – Update Student Data



Status: 200 OK Size: 82 Bytes Time: 9 ms

```

1 {
2   "success": true,
3   "data": {
4     "id": 9,
5     "name": "Will Wheaton",
6     "age": 28,
7     "course": "Theatre"
8   }
9 }

```

Response Headers Cookies Results Docs

```

1 {
2   "success": true,
3   "data": {
4     "id": 9,
5     "name": "Will Wheaton",
6     "age": 28,
7     "course": "Theatre"
8   }
9 }

```

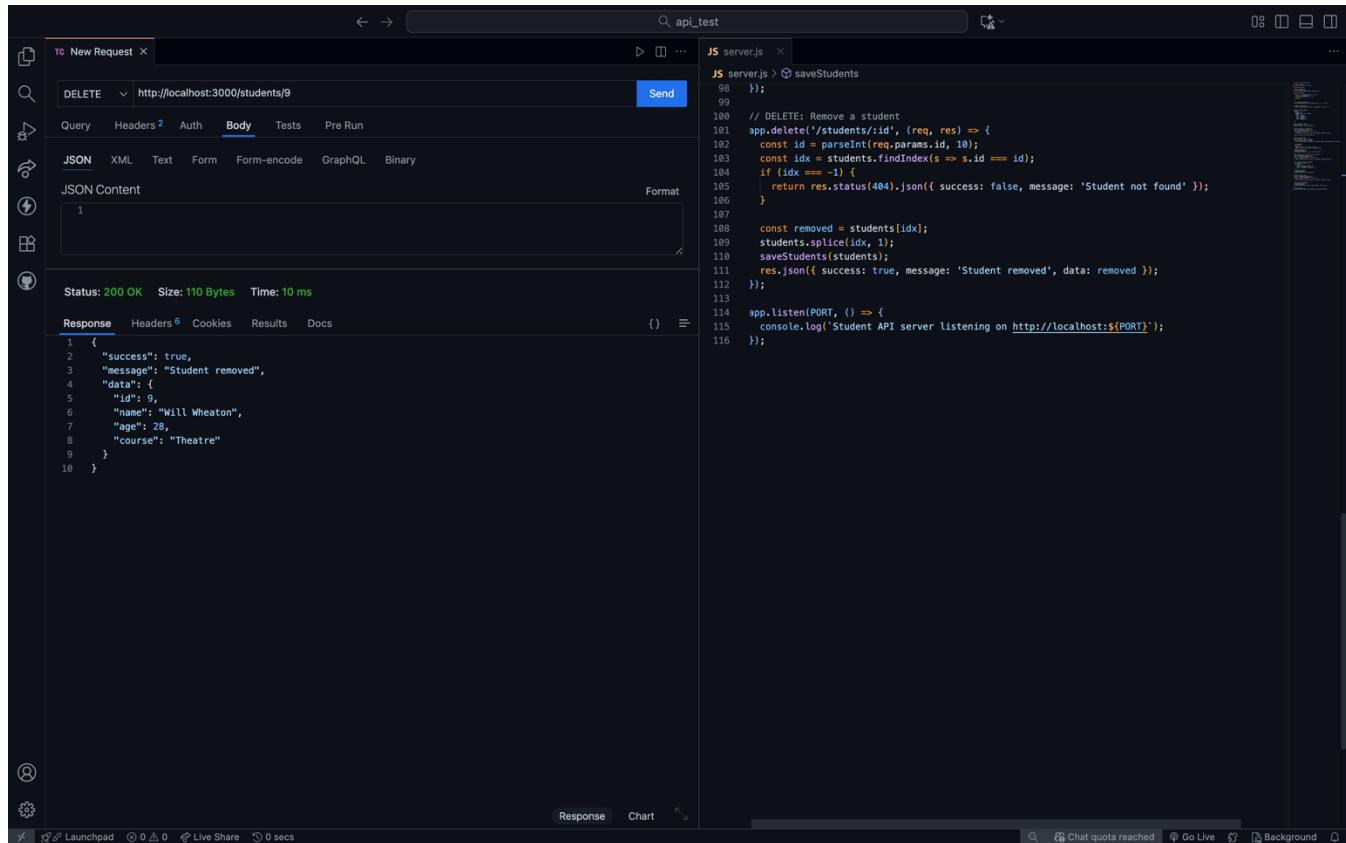
JS server.js x

```

16 const DATA_PATH = path.join(__dirname, 'students.json');
17
18 function loadStudents() {
19   try {
20     const raw = fs.readFileSync(DATA_PATH, 'utf-8');
21     const data = JSON.parse(raw);
22     return Array.isArray(data) ? data : [];
23   } catch (e) {
24     return [];
25   }
26 }
27
28 function saveStudents(students) {
29   fs.writeFileSync(DATA_PATH, JSON.stringify(students, null, 2), 'utf-8');
30 }
31
32 let students = loadStudents();
33 let nextId = students.length ? Math.max(...students.map(s => s.id || 0)) + 1 : 1;
34
35 // Root route: quick status/info
36 app.get('/', (req, res) => {
37   res.json({
38     success: true,
39     message: 'Student API is running',
40     endpoints: [
41       'GET /students',
42       'GET /students/:id',
43       'POST /students',
44       'PUT /students/:id',
45       'DELETE /students/:id'
46     ]
47   });
48 })
49
50 // GET: Retrieve all students
51 app.get('/students', (req, res) => {
52   res.json({ success: true, data: students });
53 })
54
55 // GET by ID: Retrieve a single student
56 app.get('/students/:id', (req, res) => {
57   const id = parseInt(req.params.id, 10);
58   const student = students.find(s => s.id === id);
59   if (!student) {
60     return res.status(404).json({ success: false, message: 'Student not found' });
61   }
62   res.json({ success: true, data: student });
63 })
64
65 // POST: Add a new student
66 app.post('/students', (req, res) => {
67   const { name, age, course } = req.body;
68   if (!name || typeof name !== 'string') {
69     return res.status(400).json({ success: false, message: 'Name is required and must be a string' });
70   }
71
72   const newStudent = {
73     id: nextId++,
74     name,
75     age: typeof age === 'number' ? age : undefined,
76     course: typeof course === 'string' ? course : undefined
77   };
78   students.push(newStudent);
79   saveStudents(students);
80   res.status(201).json({ success: true, data: newStudent });
81 })
82
83 // PUT: Update student details (partial update supported)
84 app.put('/students/:id', (req, res) => {
85   const id = parseInt(req.params.id, 10);
86   const student = students.find(s => s.id === id);
87   if (!student) {
88     return res.status(404).json({ success: false, message: 'Student not found' });
89   }
90
91   const updatedStudent = {
92     id: student.id,
93     name: req.body.name || student.name,
94     age: req.body.age || student.age,
95     course: req.body.course || student.course
96   };
97   students[students.indexOf(student)] = updatedStudent;
98   saveStudents(students);
99   res.status(200).json({ success: true, data: updatedStudent });
100 })
101
102 // DELETE: Remove a student
103 app.delete('/students/:id', (req, res) => {
104   const id = parseInt(req.params.id, 10);
105   const studentIndex = students.findIndex(s => s.id === id);
106   if (studentIndex === -1) {
107     return res.status(404).json({ success: false, message: 'Student not found' });
108   }
109   students.splice(studentIndex, 1);
110   saveStudents(students);
111   res.status(200).json({ success: true, message: 'Student removed' });
112 })
113
114 // Error handling
115 app.use((err, req, res, next) => {
116   console.error(err.message);
117   res.status(500).json({ success: false, message: 'Internal Server Error' });
118 })
119
120 // Listen for requests
121 app.listen(3000, () => {
122   console.log('Student API is running on port 3000');
123 })

```

DELETE Request – Remove Student Record



The screenshot shows a Thunder Client interface with two tabs: "TC New Request" and "JS server.js". The "TC New Request" tab displays a DELETE request to "http://localhost:3000/students/9". The "server.js" tab shows the Node.js code for handling the DELETE operation, which removes student ID 9 from an array of students and returns a success message.

```

TC New Request x
DELETE http://localhost:3000/students/9
Send
Query Headers 2 Auth Body Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content Format
1
Status: 200 OK Size: 110 Bytes Time: 10 ms
Response Headers Cookies Results Docs
1 {
2   "success": true,
3   "message": "Student removed",
4   "data": {
5     "id": 9,
6     "name": "Will Wheaton",
7     "age": 28,
8     "course": "Theatre"
9   }
10 }

JS server.js x
JS server.js > saveStudents
98
99
100 // DELETE: Remove a student
101 app.delete('/students/:id', (req, res) => {
102   const id = parseInt(req.params.id, 10);
103   const idx = students.findIndex(s => s.id === id);
104   if (idx === -1) {
105     return res.status(404).json({ success: false, message: 'Student not found' });
106   }
107
108   const removed = students[idx];
109   students.splice(idx, 1);
110   saveStudents(students);
111   res.json({ success: true, message: 'Student removed', data: removed });
112 });
113
114 app.listen(PORT, () => {
115   console.log(`Student API server listening on http://localhost:${PORT}`);
116 });

```

Conclusion:

In this experiment, a RESTful API was successfully implemented using Express.js and Node.js. The API handled CRUD operations effectively and was tested using Thunder Client. All endpoints returned the expected responses, demonstrating the core functionality of RESTful services.