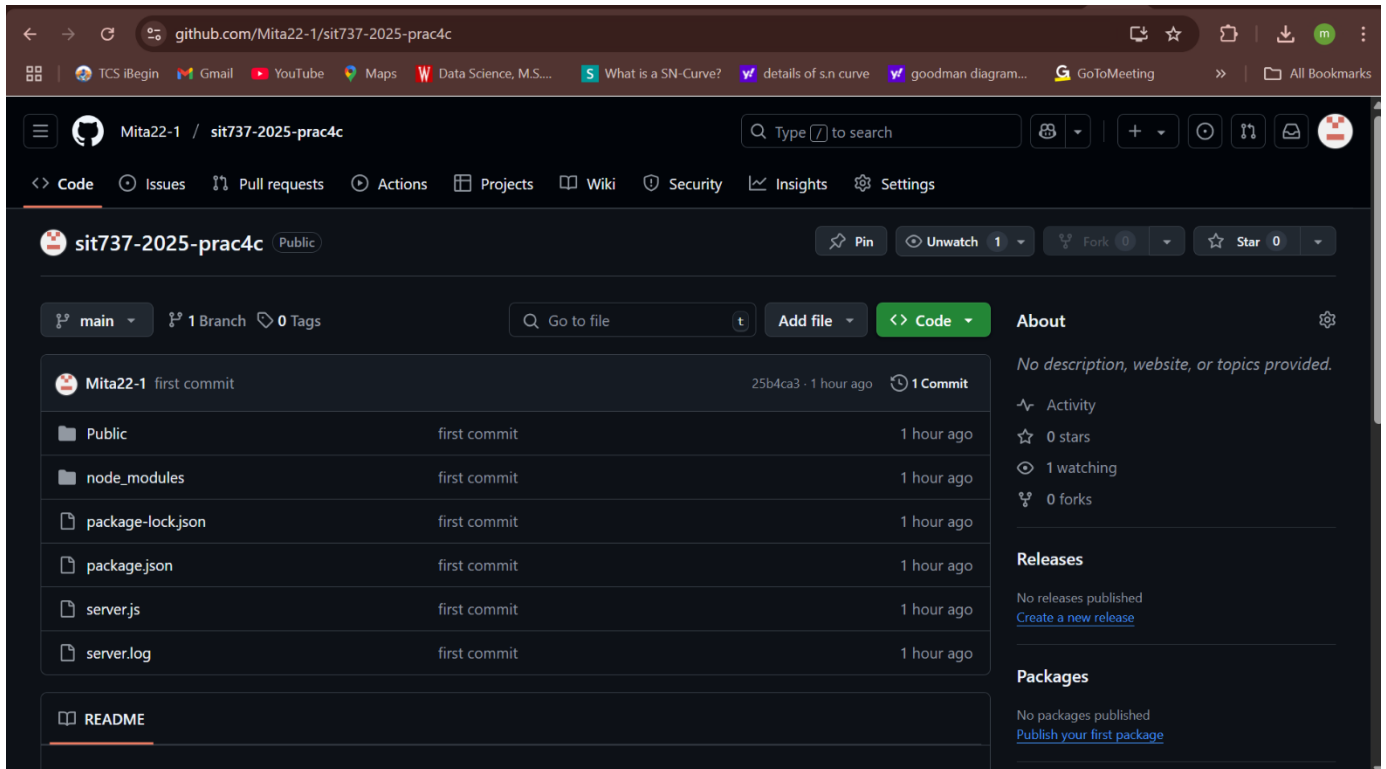


# CLOUD NATIVE APPLICATION DEVELOPMENT

By-Mitali Kaur  
(Deakin Id- s224582251)

Github Link:

<https://github.com/Mita22-1/sit737-2025-prac4c>



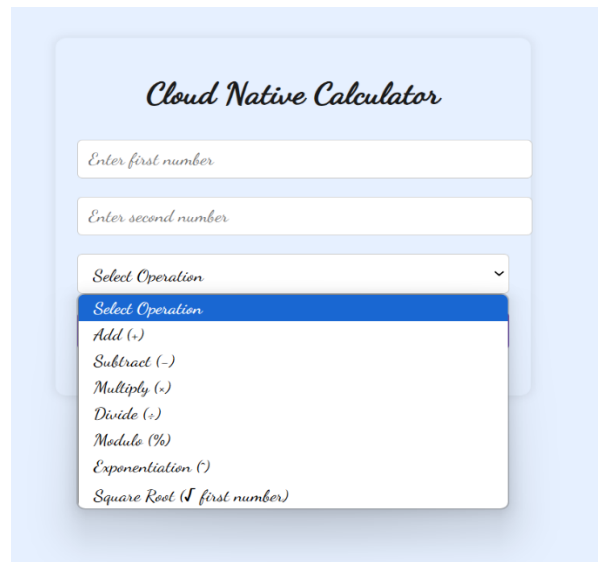
## Calculator Microservice- What It's About

This project builds upon the calculator microservice introduced earlier by expanding its functionality and improving robustness. Implemented using Node.js and Express, the microservice now supports additional arithmetic operations and adopts advanced error handling strategies, making it more reliable and practical for real-world use.

The calculator now supports:

- **Exponentiation** (/power route): Raises one number to the power of another.
- **Square Root** (/sqrt route): Computes the square root of a number.
- **Modulo** (/modulo route): Returns the remainder of division between two numbers.

Each operation is accessible through a dedicated API endpoint, allowing both UI-based and programmatic interaction.

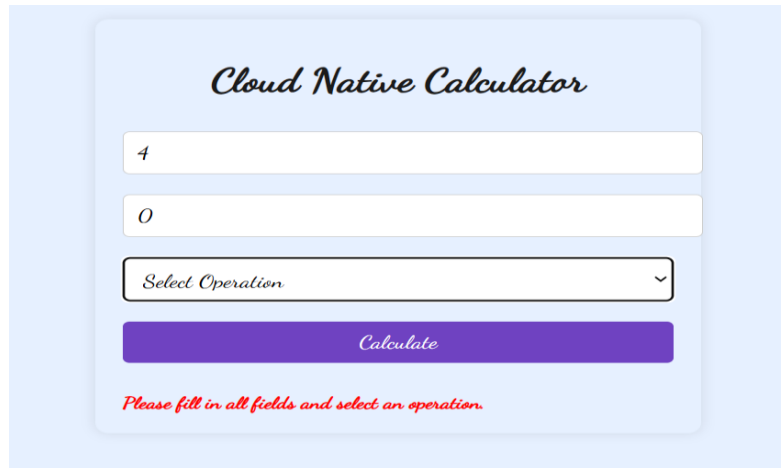


The image shows a web interface for a "Cloud Native Calculator". It features a light blue background with a white rounded rectangle containing the calculator controls. At the top, the title "Cloud Native Calculator" is written in a cursive font. Below the title are two input fields: "Enter first number" and "Enter second number". Under these is a dropdown menu labeled "Select Operation" with a downward arrow. The dropdown menu is open, showing a list of operations: "Add (+)", "Subtract (-)", "Multiply (\*)", "Divide (/)", "Module (%)", "Exponentiation (^)", and "Square Root (√ first number)".

## Handling Errors & Keeping Logs

Robust error handling mechanisms ensure invalid inputs, such as non-numeric values or illegal operations (e.g., dividing by zero or square root of a negative number), are caught and handled gracefully.

- **Winston Logger** tracks every request and error, supporting monitoring and debugging.



*Cloud Native Calculator*

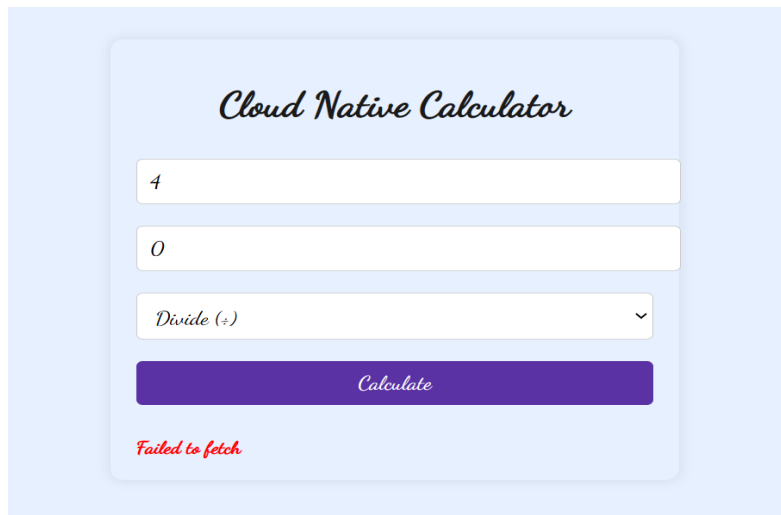
4

0

Select Operation ▾

Calculate

*Please fill in all fields and select an operation.*



*Cloud Native Calculator*

4

0

Divide (÷) ▾

Calculate

*Failed to fetch.*

**Logging:**

# Index.html

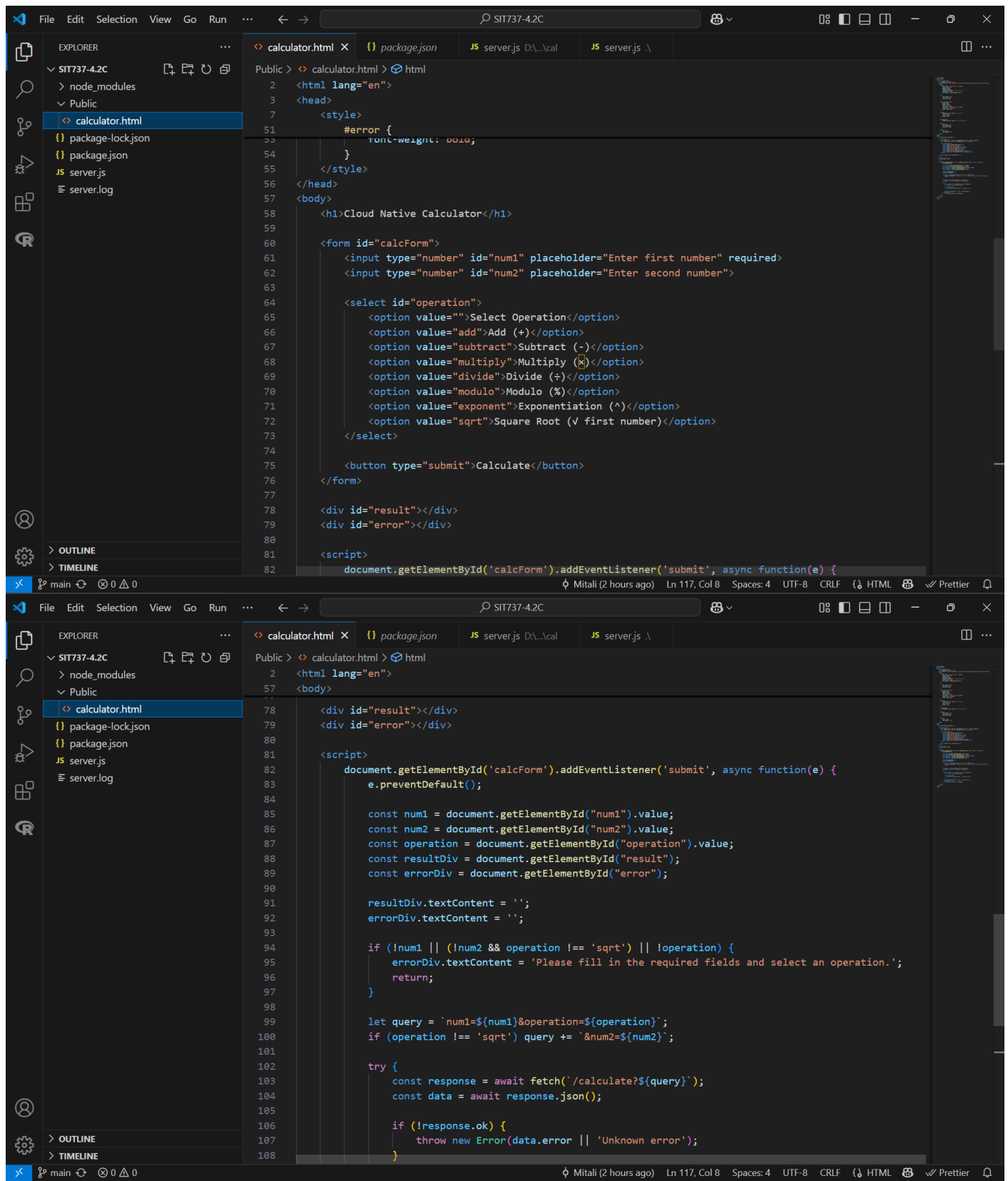
The image displays two screenshots of a Visual Studio Code editor window, showing the development of an HTML file named `calculator.html`. The Explorer sidebar on the left shows the project structure, including `package-lock.json`, `package.json`, `server.js`, and `server.log`.

**Top Screenshot:** The editor shows the initial HTML structure and CSS for a light blue box. The HTML includes a DOCTYPE declaration, a title "Elegant Calculator", and a link to a Google Font. The CSS defines a box with a light blue background, a border, and a shadow. It also sets the font family to 'Dancing Script' and defines styles for `h1` and `input, select, button` elements.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Elegant Calculator</title>
6   <link href="https://fonts.googleapis.com/css2?family=Dancing+Script&display=swap" rel="stylesheet">
7   <style>
8     body {
9       font-family: 'Dancing Script', cursive;
10      margin: 40px auto;
11      padding: 20px;
12      max-width: 400px;
13      background: #e6f0ff; /* Light blue box */
14      border-radius: 10px;
15      box-shadow: 0 0 10px rgba(0,0,0,0.1);
16    }
17
18    h1 {
19      text-align: center;
20      color: #1a1a1a;
21      font-size: 2rem;
22    }
23
24    input, select, button {
25      width: 100%;
26      padding: 10px;
27      margin: 10px 0;
28      font-size: 1.1rem;
29      font-family: 'Dancing Script', cursive;
30      border: 1px solid #ccc;
31      border-radius: 5px;
32    }
33  </style>
34  </head>
```

**Bottom Screenshot:** The editor shows the continuation of the CSS, adding styles for a purple button and a result display area. The CSS includes a purple button with a white cursor, a hover effect, and a result display area with a bold font and a specific color. It also adds an error display area with a red color and bold font.

```
35    button {
36      background-color: #6f42c1; /* Purple */
37      color: white;
38      cursor: pointer;
39      border: none;
40    }
41
42    button:hover {
43      background-color: #5a32a3; /* Darker purple */
44    }
45
46    #result {
47      font-weight: bold;
48      color: #2c3e50;
49      margin-top: 20px;
50    }
51
52    #error {
53      color: red;
54      font-weight: bold;
55    }
56  </style>
57 </head>
58 <body>
59   <h1>Cloud Native Calculator</h1>
```



The screenshot shows a Visual Studio Code editor window with a project named "SIT737-4.2C". The Explorer sidebar on the left shows the file structure: "node\_modules", "Public" (containing "calculator.html"), "package-lock.json", "package.json", "server.js", and "server.log". The main editor area displays the "calculator.html" file, which contains HTML and JavaScript code. The code defines a submit event listener for a form with ID "calcForm", which uses the Fetch API to send a POST request to "/calculate" with query parameters. It handles the response, updates the "resultDiv" with the result, and catches any errors to display in "errorDiv".

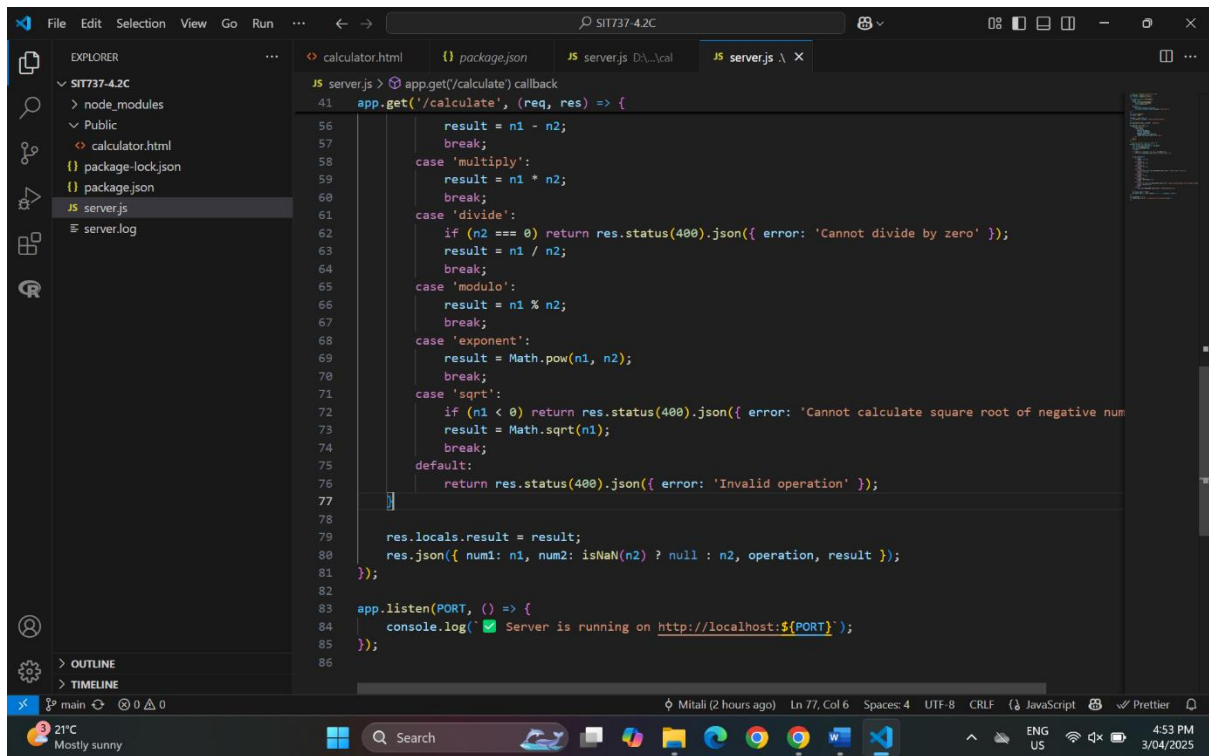
```
2 <html lang="en">
57 <body>
81   <script>
82     document.getElementById('calcForm').addEventListener('submit', async function(e) {
98
99       let query = `num1=${num1}&operation=${operation}`;
100       if (operation !== 'sqrt') query += `&num2=${num2}`;
101
102       try {
103         const response = await fetch(`/calculate?${query}`);
104         const data = await response.json();
105
106         if (!response.ok) {
107           throw new Error(data.error || 'Unknown error');
108         }
109
110         resultDiv.textContent = `Result: ${data.result}`;
111       } catch (err) {
112         errorDiv.textContent = err.message;
113       }
114     });
115   </script>
116 </body>
117 </html>
```

The status bar at the bottom indicates the current file is "main", the cursor is at line 117, column 8, and the editor is using UTF-8 encoding with CRLF line endings. It also shows that Prettier is installed and active.

# Server.js

```
1 // server.js - Updated for SIT737 4.2C Task
2 const express = require('express');
3 const winston = require('winston');
4
5 const logger = winston.createLogger({
6   level: 'info',
7   format: winston.format.combine(
8     winston.format.timestamp(),
9     winston.format.json()
10  ),
11  transports: [
12    new winston.transports.Console(),
13    new winston.transports.File({ filename: 'server.log' })
14  ]
15 });
16
17 const app = express();
18 const PORT = 3111;
19
20 app.get('/', (req, res) => {
21   res.sendFile(__dirname + '/public/calculator.html');
22 });
23
24 app.use(express.static(__dirname + '/public'));
25
26 app.use((req, res, next) => {
27   res.on('finish', () => {
28     logger.info({
29       ip: req.ip,
30       method: req.method,
31       url: req.originalUrl,
32       headers: req.headers,
33       responseStatus: res.statusCode,
34       result: res.locals.result || 'No result'
35     });
36   });
37   next();
38 });
39
40 // Enhanced Calculator logic for SIT737-4.2C
41 app.get('/calculate', (req, res) => {
42   const { num1, num2, operation } = req.query;
43   const n1 = parseFloat(num1);
44   const n2 = parseFloat(num2);
45   let result;
46
47   if (isNaN(n1) || (operation !== 'sqrt' && isNaN(n2))) {
48     return res.status(400).json({ error: 'Invalid numbers' });
49   }
50
51   switch (operation) {
52     case 'add':
53       result = n1 + n2;
54       break;
55     case 'subtract':
56       result = n1 - n2;
57       break;
58     case 'multiply':
59       result = n1 * n2;
60       break;
61     case 'divide':
62       if (n2 === 0) return res.status(400).json({ error: 'Cannot divide by zero' });
63       result = n1 / n2;
64   }
65
66   res.json({ result });
67 });
```





```
JS server.js > app.get('/calculate') callback
41 app.get('/calculate', (req, res) => {
56     result = n1 - n2;
57     break;
58     case 'multiply':
59     result = n1 * n2;
60     break;
61     case 'divide':
62     if (n2 === 0) return res.status(400).json({ error: 'Cannot divide by zero' });
63     result = n1 / n2;
64     break;
65     case 'modulo':
66     result = n1 % n2;
67     break;
68     case 'exponent':
69     result = Math.pow(n1, n2);
70     break;
71     case 'sqrt':
72     if (n1 < 0) return res.status(400).json({ error: 'Cannot calculate square root of negative num' });
73     result = Math.sqrt(n1);
74     break;
75     default:
76     return res.status(400).json({ error: 'Invalid operation' });
77 }
78
79 res.locals.result = result;
80 res.json({ num1: n1, num2: isNaN(n2) ? null : n2, operation, result });
81 });
82
83 app.listen(PORT, () => {
84     console.log(' Server is running on http://localhost:${PORT}');
85 });
86
```

## Wrapping Up

This project started as a basic calculator but grew into something more powerful. With new features like square root, exponentiation, and modulo, it now handles more than just the basics. I also looked into how microservices deal with errors, learning about smart patterns like circuit breakers and retries.

Overall, this was a great way to get hands-on with building better, more reliable services—and there's still room to grow it further.