Task 4.2C

Calculator Microservice – Documentation

This document provides a **step-by-step guide** on setting up, running, and understanding the **Calculator Microservice** built using **Node.js** and **Express.js**. The microservice includes **four arithmetic operations** (addition, subtraction, multiplication, division), and three other operations such as, exponentiation, square root and modulus, and **logs requests and errors** using **Winston**.

1. **Project Overview**

   The **Calculator Microservice** is a **RESTful API** that performs basic arithmetic operations. It is implemented using:

   - **Node.js & Express.js** – Handles API requests
   - **Winston** – Logs all API calls and errors
   - **GitHub** – Stores the source code for collaboration

2. **Project Folder Structure**

   calculator-microservice/
   |-- logs/ <-- Stores log files
   | ├── combined.log <-- Logs all API requests
   | ├── error.log <-- Logs errors only
   |-- server.js <-- Main API code
   |-- package.json <-- Project dependencies and metadata
   |-- node_modules/ <-- Installed dependencies

3. **Step-by-Step Setup Guide**

   - Step 1: Install Node.js
     Verify **Node.js** is installed using:
     - *node –v*
     - *npm –v*

   - Step 2: Initialize the Node.js Project
     Navigate to the project folder and initialize a Node.js project:
     - *npm init –y*

     This creates a *package.json* file.

   - Step 3: Install Dependencies
     Install the required packages:
     - *npm install express Winston*

     express – For handling HTTP requests
     winston – For logging API requests and errors

4. API Implementation (server.js)

- Import Required Modules

```
const express = require('express');
const winston = require('winston');
```

express – Creates the API server
winston – Logs API requests and errors

- Initialize Express App

```
const app = express();
const PORT = 3000;
```

app – Stores the Express application
PORT – Defines the port number for the server

- Configure Winston Logging

```
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  defaultMeta: { service: 'calculator-microservice' },
  transports: [
    new winston.transports.Console({ format: winston.format.simple() }),
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.File({ filename: 'logs/combined.log' }),
  ],
});
```

Console Transport – Displays logs in the terminal
File Transports – Saves logs in *logs/error.log* and *logs/combined.log*

- Middleware to Log Incoming Requests

```
app.use((req, res, next) => {
  logger.info(`Received ${req.method} request for ${req.url}`);
  next();
});
```

Logs each request to *combined.log*

- API Endpoints
  Each endpoint takes **two query parameters** (*num1* and *num2*), performs the respective operation, and returns the result.

  - Addition Endpoint

```
app.get('/add', (req, res) => {
  const num1 = parseFloat(req.query.num1);
  const num2 = parseFloat(req.query.num2);
  if (isNaN(num1) || isNaN(num2)) {
    logger.error('Invalid input for addition');
    return res.status(400).json({ error: 'Invalid numbers' });
  }
  const result = num1 + num2;
  res.json({ operation: 'addition', result });
});
```

    Validates numbers
    Adds two numbers
    Returns JSON response

  - Subtraction Endpoint

```
app.get('/subtract', (req, res) => {
  const num1 = parseFloat(req.query.num1);
  const num2 = parseFloat(req.query.num2);
  if (isNaN(num1) || isNaN(num2)) {
    logger.error('Invalid input for subtraction');
    return res.status(400).json({ error: 'Invalid numbers' });
  }
  const result = num1 - num2;
  res.json({ operation: 'subtraction', result });
});
```

    Validates numbers
    Subtracts two numbers
    Returns JSON response

  - Multiplication Endpoint

```
app.get('/multiply', (req, res) => {
  const num1 = parseFloat(req.query.num1);
  const num2 = parseFloat(req.query.num2);
  if (isNaN(num1) || isNaN(num2)) {
    logger.error('Invalid input for multiplication');
```

```
    return res.status(400).json({ error: 'Invalid numbers' });
  }
  const result = num1 * num2;
  res.json({ operation: 'multiplication', result });
});
```

Validates numbers
Multiply two numbers
Returns JSON response

- Division Endpoint

```
app.get('/divide', (req, res) => {
  const num1 = parseFloat(req.query.num1);
  const num2 = parseFloat(req.query.num2);
  if (isNaN(num1) || isNaN(num2) || num2 === 0) {
    logger.error('Invalid input for division');
    return res.status(400).json({ error: 'Invalid numbers or division by zero' });
  }
  const result = num1 / num2;
  res.json({ operation: 'division', result });
});
```

Validates numbers
Checks for division by zero
Divides two numbers
Returns JSON response

- Exponentiation

```
app.get('/power', (req, res) => {
  const { num1, num2 } = req.query;
  if (!num1 || !num2 || isNaN(num1) || isNaN(num2)) {
    logger.error('Invalid input for exponentiation');
    return res.status(400).json({ error: 'Invalid input' });
  }
  const result = Math.pow(parseFloat(num1), parseFloat(num2));
  logger.info(`Exponentiation: ${num1}^${num2} = ${result}`);
  res.json({ result });
});
```

Validates numbers
Takes power of $num1^{num2}$
Returns JSON response

- Square Root

```
app.get('/sqrt', (req, res) => {
  const { num1 } = req.query;
  if (!num1 || isNaN(num1) || num1 < 0) {
    logger.error('Invalid input for square root');
    return res.status(400).json({ error: 'Invalid input, number must be non-negative' });
  }
  const result = Math.sqrt(parseFloat(num1));
  logger.info(`Square Root: √${num1} = ${result}`);
  res.json({ result });
});
```

This is the only endpoint with only **one query parameter**.
Validates numbers
Takes square root of number 1
Returns JSON response

- Modulo

```
app.get('/modulo', (req, res) => {
  const { num1, num2 } = req.query;
  if (!num1 || !num2 || isNaN(num1) || isNaN(num2) || num2 == 0) {
    logger.error('Invalid input for modulo operation');
    return res.status(400).json({ error: 'Invalid input or division by zero' });
  }
  const result = parseFloat(num1) % parseFloat(num2);
  logger.info(`Modulo: ${num1} % ${num2} = ${result}`);
  res.json({ result });
});
```

Validates numbers
Takes modulus of num1 with num2, num1 % num2
Returns JSON response

- Start the Server

```
app.listen(PORT, () => {
  console.log(`Calculator microservice running at http://localhost:${PORT}`);
});
```

Starts the server on **port 3000**

Testing the services

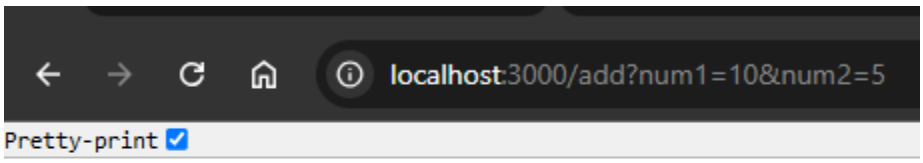
- Step 1: Start the Server

  node server.js

  Gives the output:
  Calculator microservice running at http://localhost:3000

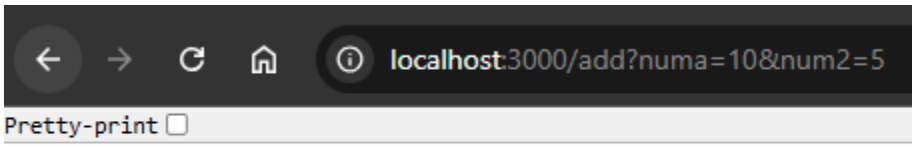
- Step 2: Test API Endpoints
  I used the browser for testing

  - Addition



Pretty-print ☑

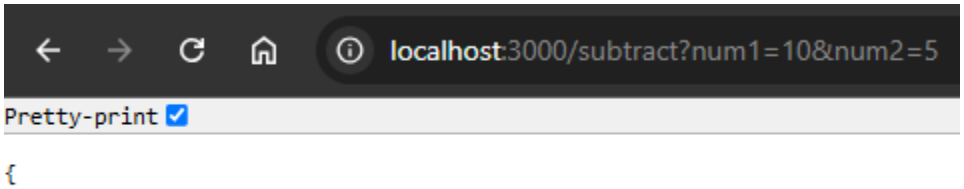
```
{
  "operation": "addition",
  "result": 15
}
```



Pretty-print ☐

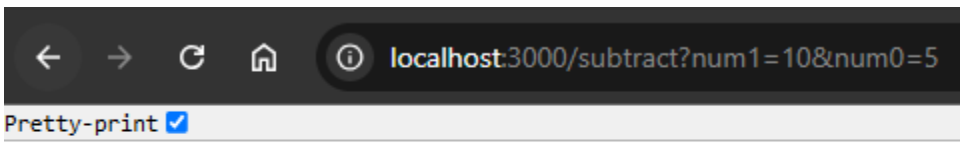
```
{"error":"Invalid input: Both num1 and num2 should be numbers."}
```

  - Subtraction



Pretty-print ☑

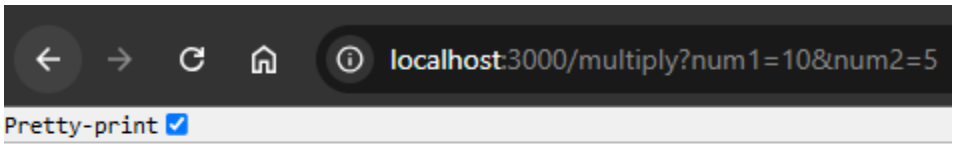
```
{
  "operation": "subtraction",
  "result": 5
}
```



Pretty-print ☑

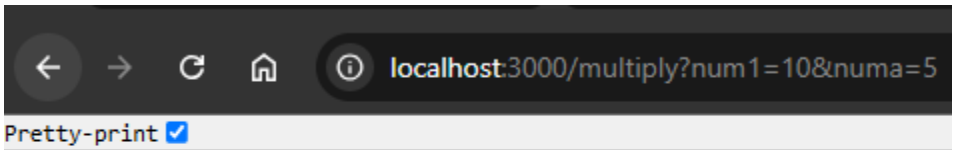
```
{
  "error": "Invalid input: Both num1 and num2 should be numbers."
}
```

- Multiplication



Pretty-print ☑

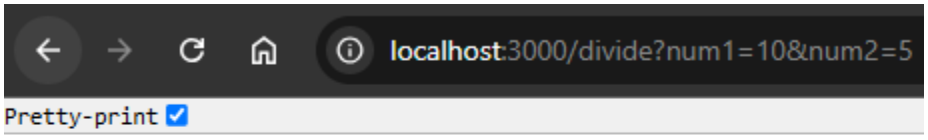

```
{
  "operation": "multiplication",
  "result": 50
}
```

Pretty-print ☑

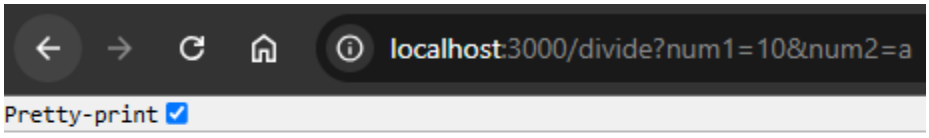

```
{
  "error": "Invalid input: Both num1 and num2 should be numbers."
}
```


- Division



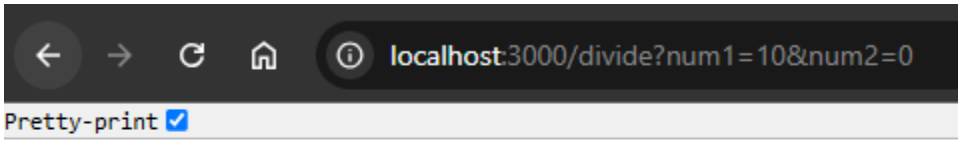
Pretty-print ☑


```
{
  "operation": "division",
  "result": 2
}
```

Pretty-print ☑

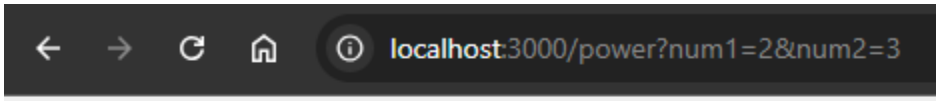

```
{
  "error": "Invalid input: Both num1 and num2 should be numbers."
}
```

Pretty-print ☑

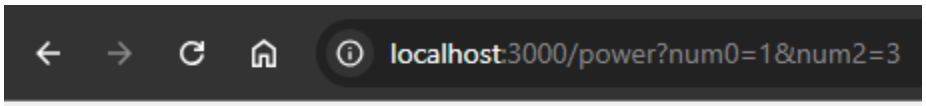

```
{
  "error": "Division by zero is not allowed."
}
```

- Exponentiation



Pretty-print ☑

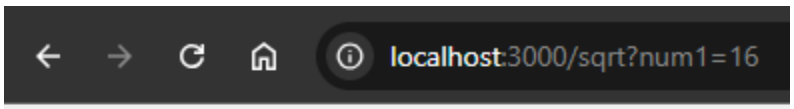

```
{
  "operation": "Exponent",
  "result": 8
}
```

Pretty-print ☑

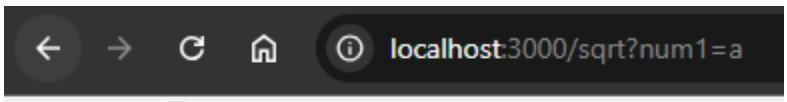

```
{
  "error": "Invalid input"
}
```


- Square Root



Pretty-print ☑

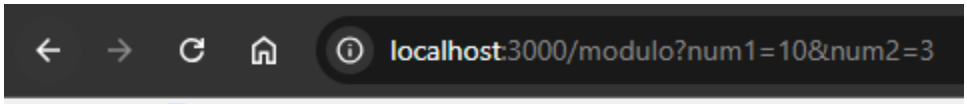

```
{
  "operation": "Square Root",
  "result": 4
}
```

Pretty-print ☑

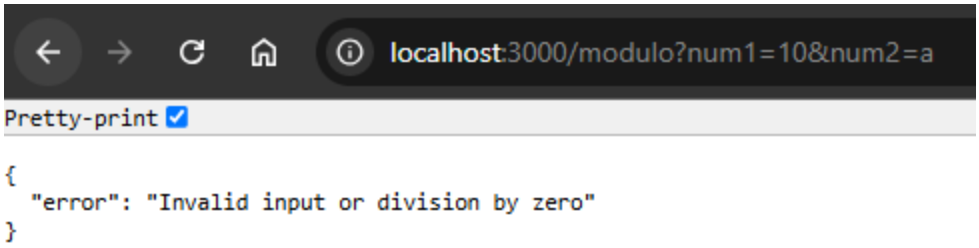

```
{
  "error": "Invalid input, number must be non-negative"
}
```


- Modulo



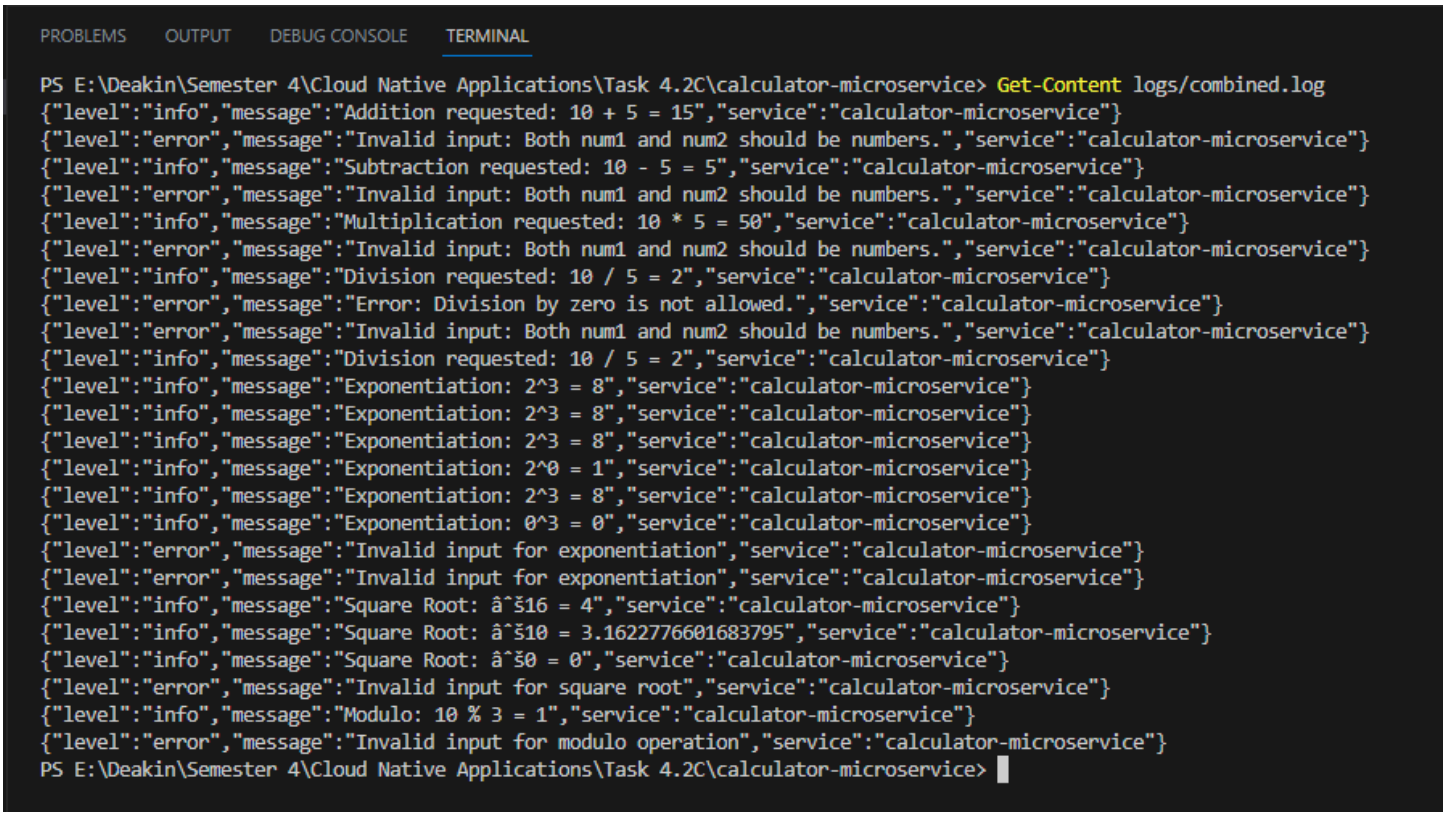
Pretty-print ☑


```
{
  "operation": "Modulo",
  "result": 1
}
```

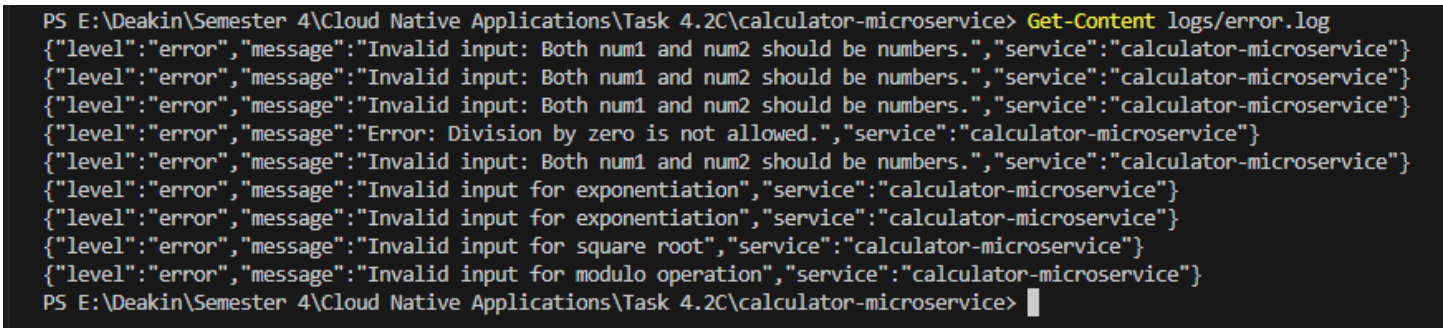


Viewing Logs

- Step 1: View Logs in Real-Time
  Run:
  Get-Content logs/combined.log



- Step 2: View Error Logs
  Run:
  Get-Content logs/error.log



Conclusion

This microservice successfully:

- Implements **addition, subtraction, multiplication, division, exponentiation, square root and modulus**
- Uses **Express.js** for API handling
- Uses **Winston** for logging
- Handles **error conditions**
- Stores logs for monitoring