4.2C - Extended Calculator Microservice – Documentation

Github Link : https://github.com/Pasindufdo98/sit737-2025-prac4c

This project builds upon the work done in Practical Task 4.1P. To start, I made a copy of the original 4.1P code and used it as the base for this extended version. The goal was to introduce a few additional mathematical operations and improve overall functionality, while still maintaining proper logging and error handling practices.

What's New

In addition to the basic operations (addition, subtraction, multiplication, and division), the microservice now supports:

- Exponentiation – calculates the result of a number raised to a power.
- Square root – finds the square root of a single number.
- Modulo – returns the remainder after dividing one number by another.

Steps Taken to Extend the Microservice

1. Copied the Code from 4.1P

I began by creating a copy of the original Practical 4.1P project files so that I could build on top of a working version without affecting the original.

2. Added New Math Functions

In the main server file, I added the logic for three new operations: exponentiation, square root, and modulo. These functions were written in a similar structure to the existing math operations.

3. Created New Endpoints

To make these new functions accessible, I created new API endpoints for each one:

/exponentiation

/square-root

/modulo

Each endpoint accepts query parameters and calls the relevant function, just like the original four operations.

4. Updated Input Validation

I updated the validation logic to handle cases where only one parameter is needed (e.g., square root), and to provide meaningful error messages if the input is invalid or missing.

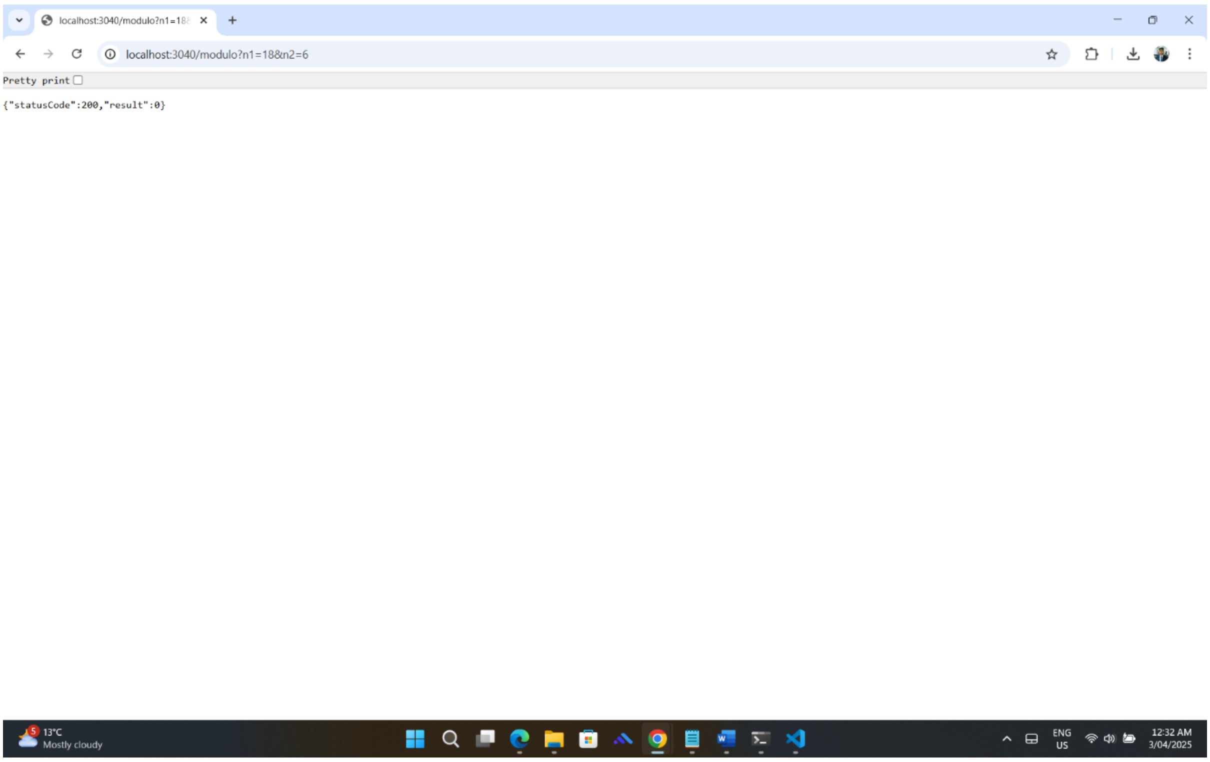
5. Maintained Error Handling

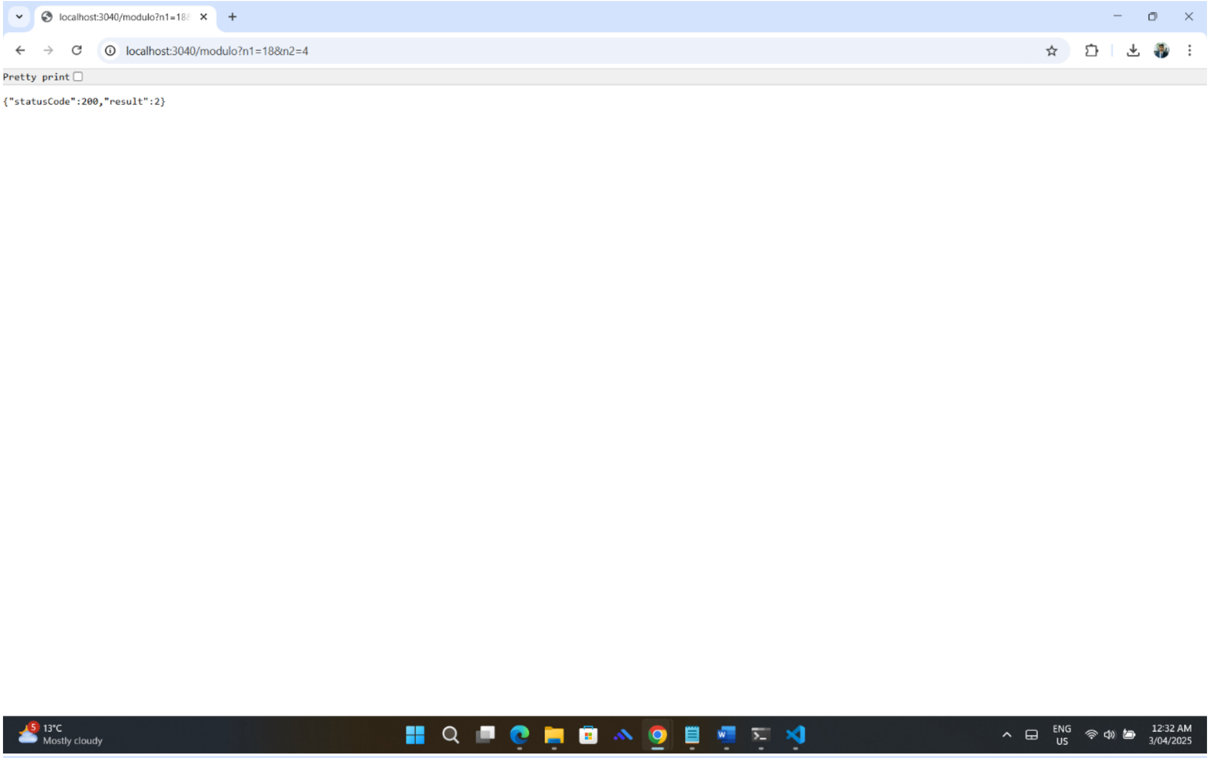
All operations were wrapped in try-catch blocks to ensure errors (like division by zero or invalid inputs) are caught and handled gracefully.

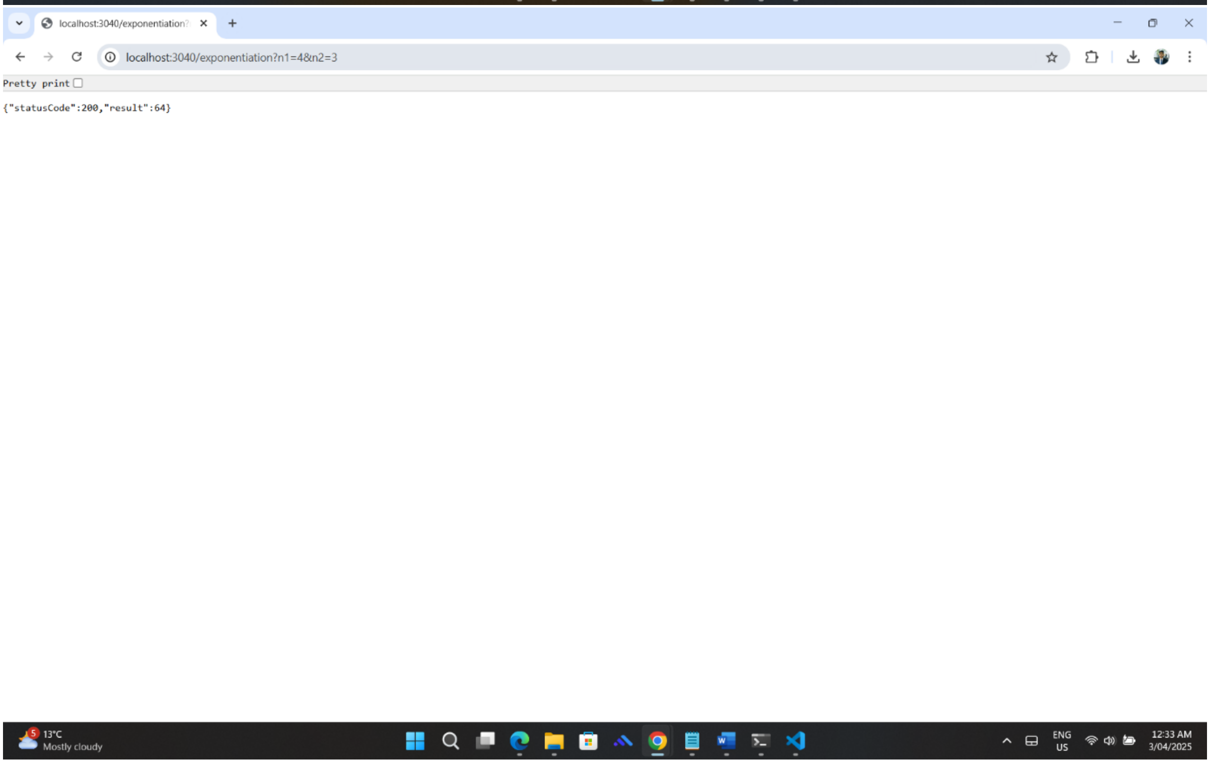
6. Continued Using Winston for Logging

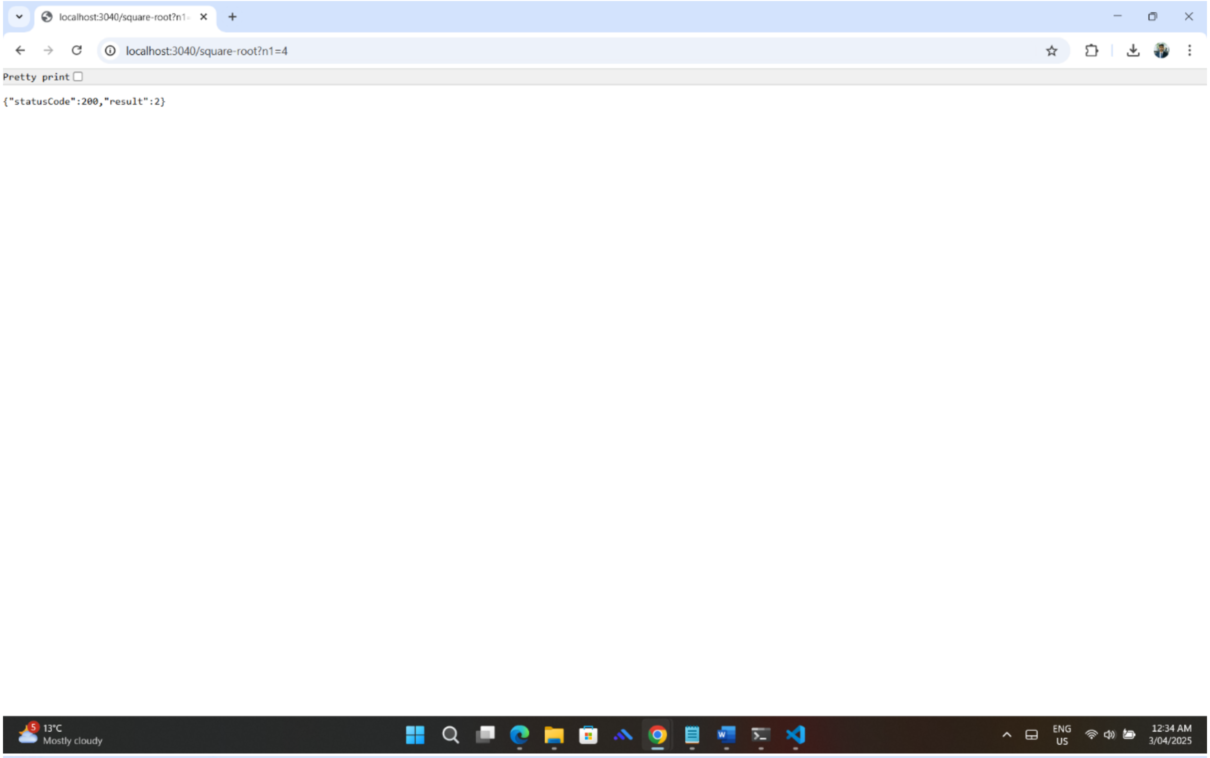
The Winston logger from 4.1P was kept and used throughout the new operations. Each request is logged with details such as input values, type of operation, and any error messages if something goes wrong.

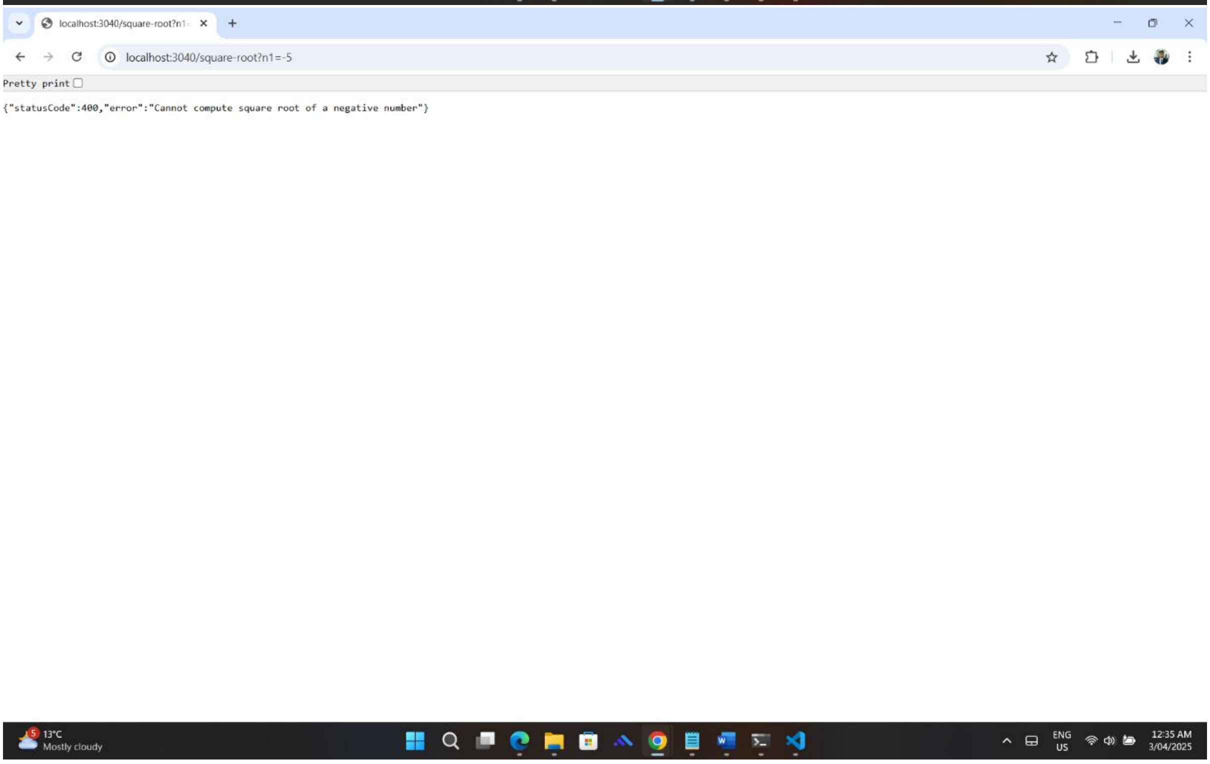
Screenshots

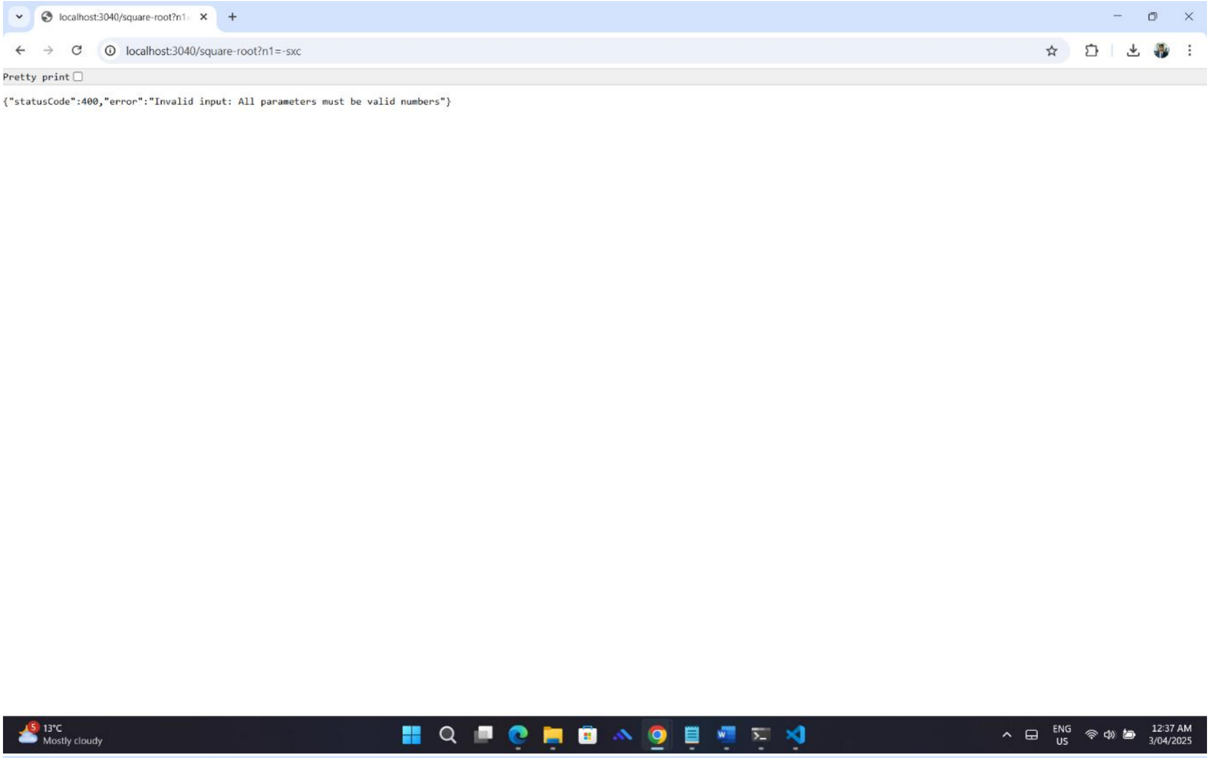
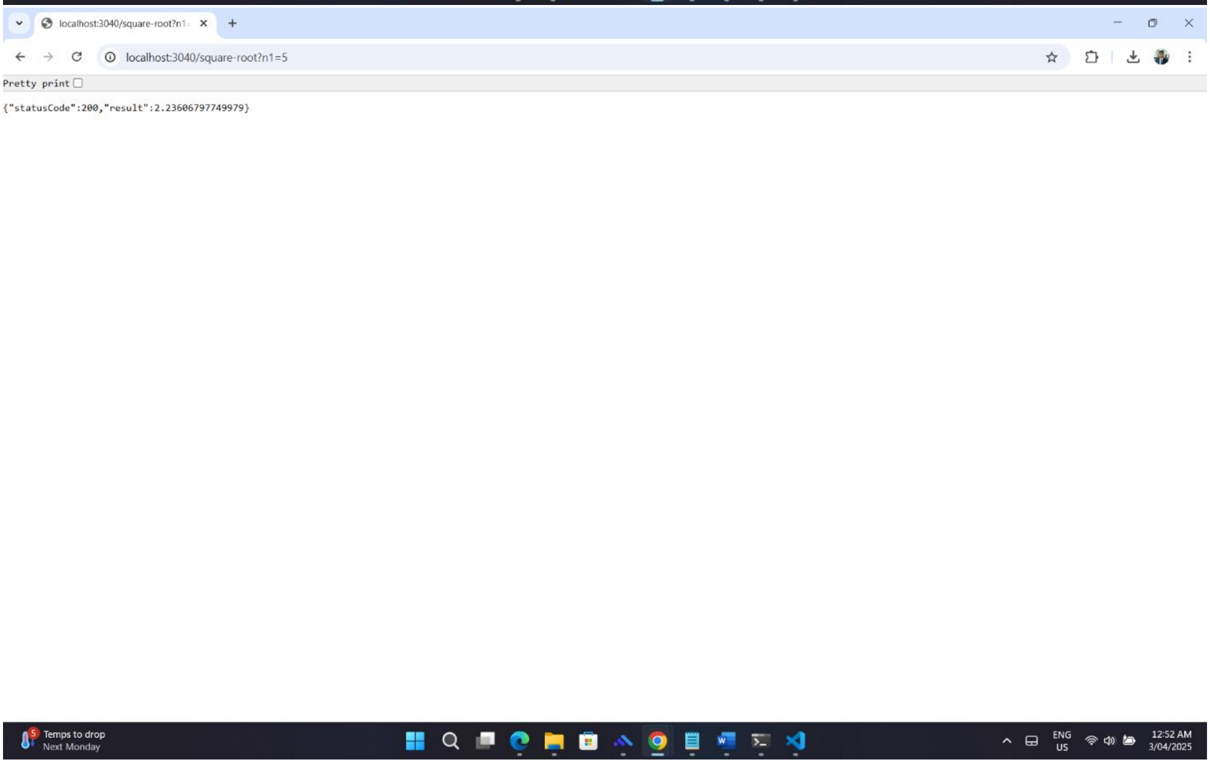


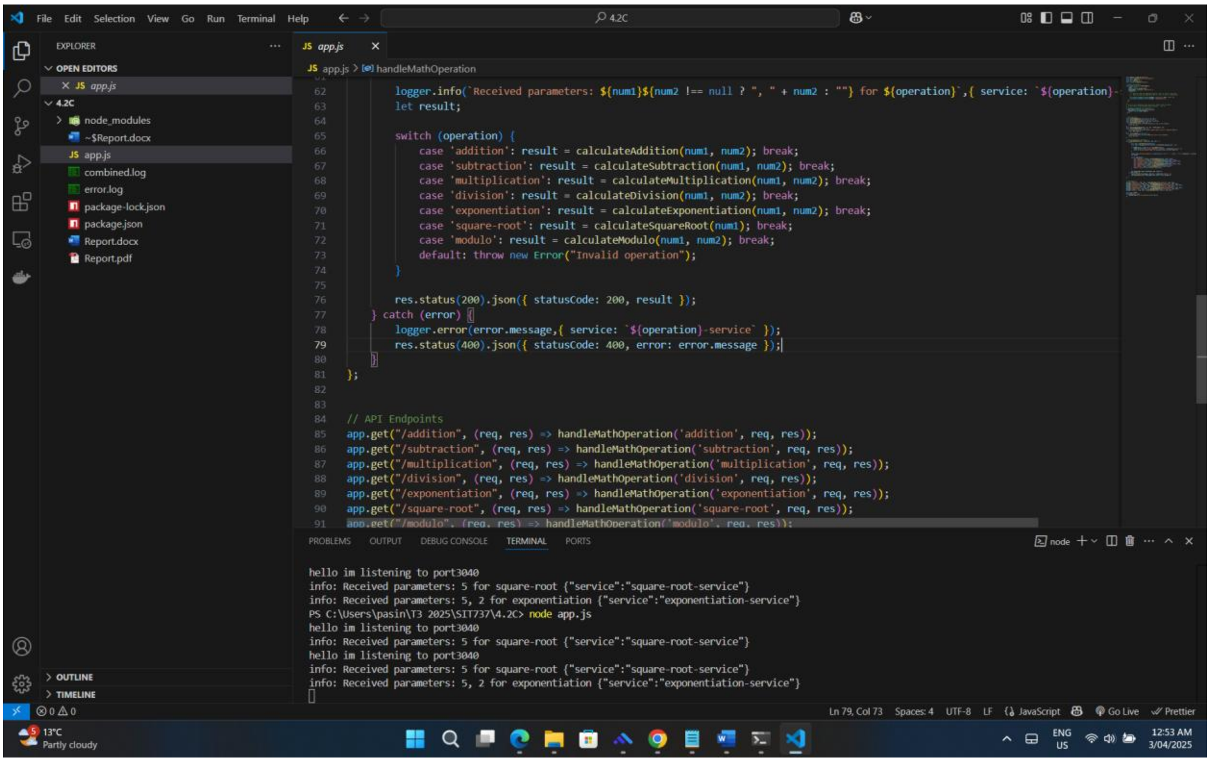

localhost:3040/modulo?n1=18&
localhost:3040/modulo?n1=18&n2=4
Pretty print
{"statusCode":200,"result":2}
13°C
Mostly cloudy
ENG
US
12:32 AM
3/04/2025

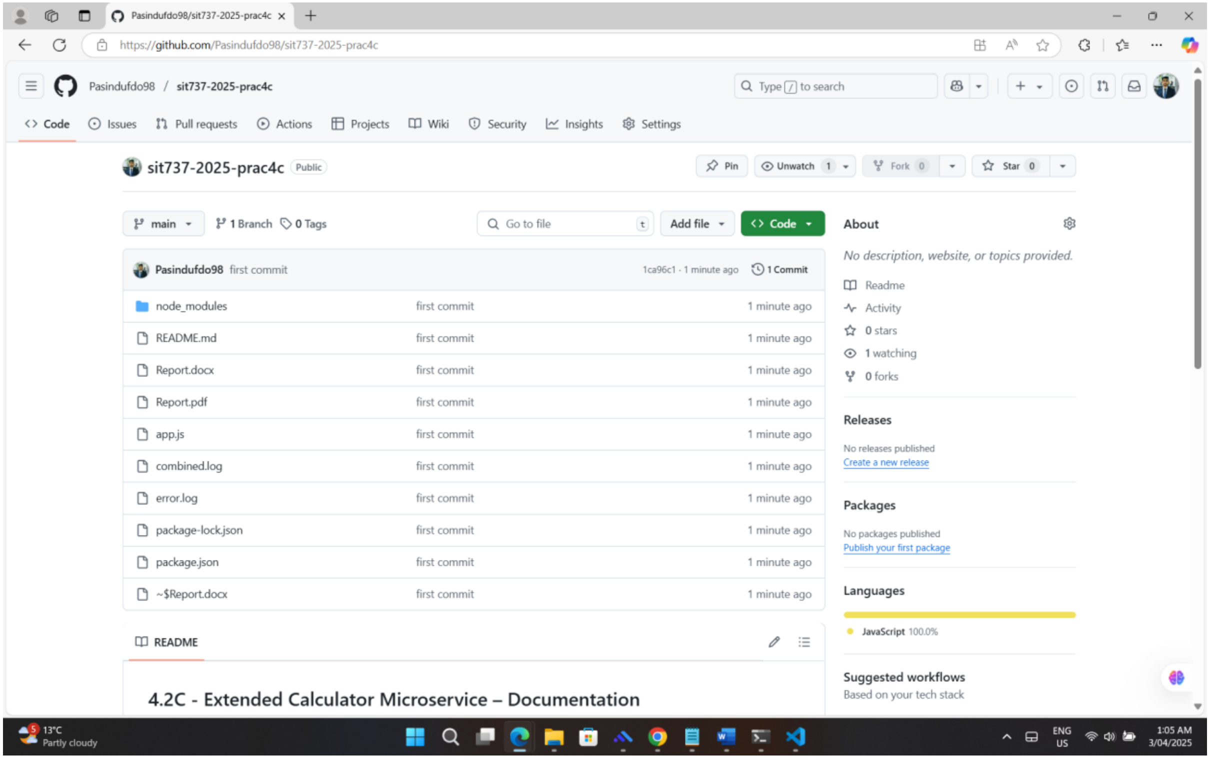

localhost:3040/exponentiation?
localhost:3040/exponentiation?n1=4&n2=3
Pretty print
{"statusCode":200,"result":64}
13°C
Mostly cloudy
ENG
US
12:33 AM
3/04/2025


localhost:3040/square-root?n1=
localhost:3040/square-root?n1=4
Pretty print
{"statusCode":200,"result":2}
13°C
Mostly cloudy
ENG
US
12:34 AM
3/04/2025


localhost:3040/square-root?n1=
localhost:3040/square-root?n1=-5
Pretty print
{"statusCode":400,"error":"Cannot compute square root of a negative number"}
13°C
Mostly cloudy
ENG
US
12:35 AM
3/04/2025


localhost:3040/square-root?n1=
localhost:3040/square-root?n1=-sxc
Pretty print
{"statusCode":400,"error":"Invalid input: All parameters must be valid numbers"}
13°C
Mostly cloudy
ENG
US
12:37 AM
3/04/2025

localhost:3040/square-root?n1=
localhost:3040/square-root?n1=5
Pretty print
{"statusCode":200,"result":2.23606797749979}
Temps to drop
Next Monday
ENG
US
12:52 AM
3/04/2025


File Edit Selection View Go Run Terminal Help
4.2C
EXPLORER
OPEN EDITORS
app.js
4.2C
node_modules
~$Report.docx
app.js
combined.log
error.log
package-lock.json
package.json
Report.docx
Report.pdf
app.js > handleMathOperation
logger.info(`Received parameters: ${num1}${num2 !== null ? ", " + num2 : ""} for ${operation}`,{ service: `${operation}-
let result;
switch (operation) {
case 'addition': result = calculateAddition(num1, num2); break;
case 'subtraction': result = calculateSubtraction(num1, num2); break;
case 'multiplication': result = calculateMultiplication(num1, num2); break;
case 'division': result = calculateDivision(num1, num2); break;
case 'exponentiation': result = calculateExponentiation(num1, num2); break;
case 'square-root': result = calculateSquareRoot(num1); break;
case 'modulo': result = calculateModulo(num1, num2); break;
default: throw new Error("Invalid operation");
}
res.status(200).json({ statusCode: 200, result });
} catch (error) {
logger.error(error.message,{ service: `${operation}-service` });
res.status(400).json({ statusCode: 400, error: error.message });
}
};
// API Endpoints
app.get("/addition", (req, res) => handleMathOperation('addition', req, res));
app.get("/subtraction", (req, res) => handleMathOperation('subtraction', req, res));
app.get("/multiplication", (req, res) => handleMathOperation('multiplication', req, res));
app.get("/division", (req, res) => handleMathOperation('division', req, res));
app.get("/exponentiation", (req, res) => handleMathOperation('exponentiation', req, res));
app.get("/square-root", (req, res) => handleMathOperation('square-root', req, res));
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
hello im listening to port3040
info: Received parameters: 5 for square-root {"service":"square-root-service"}
info: Received parameters: 5, 2 for exponentiation {"service":"exponentiation-service"}
PS C:\Users\pasin\T3 2025\SIT737\4.2C> node app.js
hello im listening to port3040
info: Received parameters: 5 for square-root {"service":"square-root-service"}
hello im listening to port3040
info: Received parameters: 5 for square-root {"service":"square-root-service"}
info: Received parameters: 5, 2 for exponentiation {"service":"exponentiation-service"}
OUTLINE
TIMELINE
Ln 79, Col 73 Spaces: 4 UTF-8 LF JavaScript Go Live Prettier
13°C Partly cloudy
ENG US 12:53 AM 3/04/2025


Pasindufdo98/sit737-2025-prac4c
https://github.com/Pasindufdo98/sit737-2025-prac4c
Pasindufdo98 / sit737-2025-prac4c
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
sit737-2025-prac4c Public
Pin Unwatch 1 Fork 0 Star 0
main 1 Branch 0 Tags
Go to file Add file Code
Pasindufdo98 first commit 1ca96c1 · 1 minute ago 1 Commit
node_modules first commit 1 minute ago
README.md first commit 1 minute ago
Report.docx first commit 1 minute ago
Report.pdf first commit 1 minute ago
app.js first commit 1 minute ago
combined.log first commit 1 minute ago
error.log first commit 1 minute ago
package-lock.json first commit 1 minute ago
package.json first commit 1 minute ago
~$Report.docx first commit 1 minute ago
README
4.2C - Extended Calculator Microservice – Documentation
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
JavaScript 100.0%
Suggested workflows
Based on your tech stack
13°C Partly cloudy
ENG US 1:05 AM 3/04/2025

README

4.2C - Extended Calculator Microservice – Documentation

This project builds upon the work done in Practical Task 4.1P. To start, I made a copy of the original 4.1P code and used it as the base for this extended version. The goal was to introduce a few additional mathematical operations and improve overall functionality, while still maintaining proper logging and error handling practices.

What's New

In addition to the basic operations (addition, subtraction, multiplication, and division), the microservice now supports:

• Exponentiation – calculates the result of a number raised to a power. • Square root – finds the square root of a single number. • Modulo – returns the remainder after dividing one number by another.

Steps Taken to Extend the Microservice

1. Copied the Code from 4.1P I began by creating a copy of the original Practical 4.1P project files so that I could build on top of a working version without affecting the original.
2. Added New Math Functions In the main server file, I added the logic for three new operations: exponentiation, square root, and modulo. These functions were written in a similar structure to the existing math operations.
3. Created New Endpoints To make these new functions accessible, I created new API endpoints for each one: /exponentiation /square-root /modulo

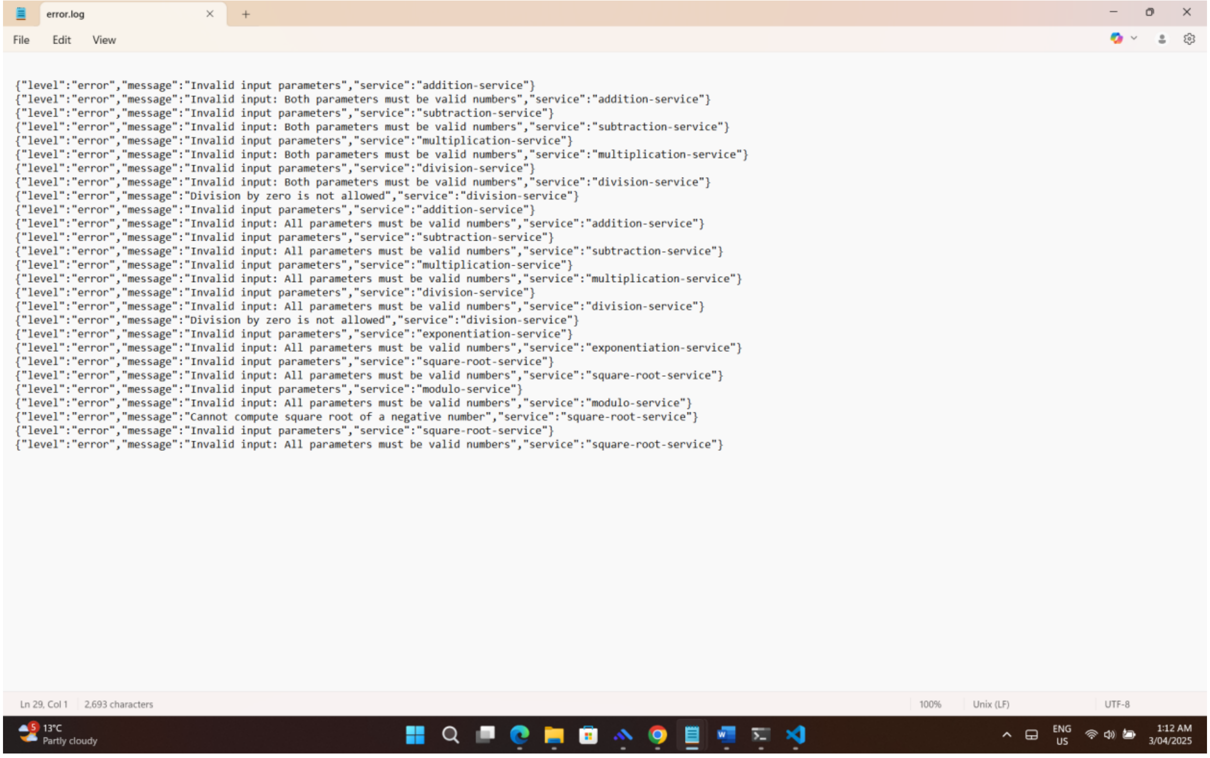
Each endpoint accepts query parameters and calls the relevant function, just like the original four operations.

4. Updated Input Validation I updated the validation logic to handle cases where only one parameter is needed (e.g., square root), and to provide meaningful error messages if the input is invalid or missing.
5. Maintained Error Handling All operations were wrapped in try-catch blocks to ensure errors (like division by zero or invalid inputs) are caught and handled gracefully.
6. Continued Using Winston for Logging The Winston logger from 4.1P was kept and used throughout the new

combined.log


```
{"level":"info","message":"Received parameters: 10, 5 for division","service":"division-service"}
{"level":"error","message":"Invalid input parameters","service":"division-service"}
{"level":"error","message":"Invalid input: Both parameters must be valid numbers","service":"division-service"}
{"level":"info","message":"Received parameters: 10, 0 for division","service":"division-service"}
{"level":"error","message":"Division by zero is not allowed","service":"division-service"}
{"level":"info","message":"Received parameters: 10, 5 for addition","service":"addition-service"}
{"level":"error","message":"Invalid input parameters","service":"addition-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"addition-service"}
{"level":"info","message":"Received parameters: 10, 5 for subtraction","service":"subtraction-service"}
{"level":"error","message":"Invalid input parameters","service":"subtraction-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"subtraction-service"}
{"level":"info","message":"Received parameters: 10, 5 for multiplication","service":"multiplication-service"}
{"level":"error","message":"Invalid input parameters","service":"multiplication-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"multiplication-service"}
{"level":"info","message":"Received parameters: 10, 5 for division","service":"division-service"}
{"level":"error","message":"Invalid input parameters","service":"division-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"division-service"}
{"level":"info","message":"Received parameters: 10, 0 for division","service":"division-service"}
{"level":"error","message":"Division by zero is not allowed","service":"division-service"}
{"level":"info","message":"Received parameters: 2, 3 for exponentiation","service":"exponentiation-service"}
{"level":"info","message":"Received parameters: 2 for exponentiation","service":"exponentiation-service"}
{"level":"error","message":"Invalid input parameters","service":"exponentiation-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"exponentiation-service"}
{"level":"info","message":"Received parameters: 25 for square-root","service":"square-root-service"}
{"level":"error","message":"Invalid input parameters","service":"square-root-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"square-root-service"}
{"level":"info","message":"Received parameters: 10, 3 for modulo","service":"modulo-service"}
{"level":"error","message":"Invalid input parameters","service":"modulo-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"modulo-service"}
{"level":"info","message":"Received parameters: 10, 5 for addition","service":"addition-service"}
{"level":"info","message":"Received parameters: 10, 7 for addition","service":"addition-service"}
{"level":"info","message":"Received parameters: 16 for square-root","service":"square-root-service"}
{"level":"info","message":"Received parameters: 64 for square-root","service":"square-root-service"}
{"level":"info","message":"Received parameters: 14, 3 for modulo","service":"modulo-service"}
{"level":"info","message":"Received parameters: 18, 6 for addition","service":"addition-service"}
{"level":"info","message":"Received parameters: 18, 6 for modulo","service":"modulo-service"}
{"level":"info","message":"Received parameters: 18, 4 for modulo","service":"modulo-service"}
{"level":"info","message":"Received parameters: 4, 3 for exponentiation","service":"exponentiation-service"}
{"level":"info","message":"Received parameters: 4 for square-root","service":"square-root-service"}
{"level":"info","message":"Received parameters: -5 for square-root","service":"square-root-service"}
{"level":"error","message":"Cannot compute square root of a negative number","service":"square-root-service"}
{"level":"error","message":"Invalid input parameters","service":"square-root-service"}
{"level":"error","message":"Invalid input: All parameters must be valid numbers","service":"square-root-service"}
{"level":"info","message":"Received parameters: 5 for square-root","service":"square-root-service"}
{"level":"info","message":"Received parameters: 5, 2 for exponentiation","service":"exponentiation-service"}
```


Ln 37, Col 94 | 5,416 characters



Part II: Various error handling strategies prevalent in microservices architecture

In microservices architecture, errors are expected due to the distributed nature of the system, so having strong error-handling strategies is essential to keep services reliable and user-friendly. One commonly used method is the Circuit Breaker pattern, which prevents a failing service from being overwhelmed by stopping calls to it once a failure threshold is reached. It switches between states “closed”, “open”, and “half-open” to test when the service is healthy again and helps avoid full system crashes. This not only protects system resources but also allows dependent services to continue functioning without being dragged down. Another supportive pattern is the Retry pattern, which automatically re-executes failed operations that might be caused by transient problems, such as network latency or timeouts. It’s important to use techniques like exponential backoff (increasing wait times) and jitter (adding randomness) to avoid retry storms that can overwhelm the network and backend systems. Developers should also ensure that operations are idempotent, meaning repeated requests won’t cause unintended effects.

Fallback mechanisms are also essential, they act as a backup by causing users to see some content even if a service is down. This can include displaying cached content, using default values, or silently masking non-critical functionality while preserving core functions. These mechanisms are especially useful in maintaining user trust and lessening frustration during partial service outages. Together, these patterns form a multi-layered defence that enables microservices to recover gracefully and fail safely. With centralized logging, real-time monitoring, and custom exception handling, they help development teams detect and resolve issues faster, leading to much less downtime. Overall, using a combination of circuit breakers, retries, and fallbacks helps build resilient microservices that can recover from failures neatly without impacting the user experience adversely, even on unforeseen failures.

References:

- MultiGenesys author, 2023. Handling Exceptions in Microservices: Best Practices. [online] MultiGenesys. Available at: https://multigenesys.com/blog/how-to-handle-exceptions-in-microservices [Accessed 31 March. 2025].
- Swift, D., 2024. Best Practices for Handling Exceptions in Java Microservices. [online] Springfuse. Available at: https://www.springfuse.com/exception-handling-best-practices-in-microservices/ [Accessed 31 March. 2025].
- Microsoft Learn, 2025. Circuit Breaker Pattern - Azure Architecture Center. [online] 22 March. Available at: https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker [Accessed 1 Apr. 2025].
- Mezo Code, 2024. Resilience in Microservices: Implementing the Retry Pattern in Java. [online] 25 June. Available at: https://mezocode.com/implementing-retry-pattern-in-java/ [Accessed 1 Apr. 2025].