

# UE23CS351B: Cloud Computing

## Lab-2: Monolithic Architecture

Name: Niharika Paul  
SRN: PES1UG23AM189

Semester: 6; Section: AIML 'D'  
Date of submission: 21-01-2026

### PART-1

```
(.venv) C:\SEM-6\CC_Lab\PES1UG23AM189\CC_Lab2>python insert_events.py
✓ Events inserted successfully!

(.venv) C:\SEM-6\CC_Lab\PES1UG23AM189\CC_Lab2>uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\SEM-6\\CC_Lab\\PES1UG23AM189\\CC_Lab2']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [4600] using StatReload
INFO:     Started server process [25840]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

### PART-2

#### SS1:

The screenshot shows a web browser window for the 'Fest Monolith' platform. The URL is 'localhost:8000/events?user=PES1UG23AM189'. The page title is 'Events'. It displays a grid of nine event cards, each with a 'Register' button. The events are:

Event ID	Event Name	Description	Price
1	Hackathon	Includes certificate • instant registration • limited seats	₹ 500
2	Dance	Includes certificate • instant registration • limited seats	₹ 300
3	Hackathon	Includes certificate • instant registration • limited seats	₹ 500
4	Dance Battle	Includes certificate • instant registration • limited seats	₹ 300
5	AI Workshop	Includes certificate • instant registration • limited seats	₹ 400
6	Photography Walk	Includes certificate • instant registration • limited seats	₹ 200
7	Gaming Tournament	Includes certificate • instant registration • limited	₹ 350
8	Music Night	Includes certificate • instant registration • limited	₹ 250
9	Treasure Hunt	Includes certificate • instant registration • limited	₹ 150

# PART-3

The screenshot shows a web browser window for the 'Fest Monolith' application at [localhost:8000/my-events?user=PES1UG23AM189](http://localhost:8000/my-events?user=PES1UG23AM189). The user is logged in as 'PES1UG23AM189'. The main content area displays a table of registered events:

Event	Fee (₹)
Photography Walk Confirmed • Fest pass	₹ 200
Music Night Confirmed • Fest pass	₹ 250
Dance Confirmed • Fest pass	₹ 300

A green button labeled 'Proceed to Checkout →' is located below the table. To the right, a 'Summary' box shows 'Total Registered 3'. A note in the summary box states: 'This page intentionally contains a backend bottleneck for performance testing using Locust.'

SS2:

The screenshot shows a web browser window for the 'Fest Monolith' application at [localhost:8000/checkout](http://localhost:8000/checkout). The user is logged in as 'PES1UG23AM189'. The main content area displays an error message: 'Monolith Failure' with a star icon, indicating 'One bug in one module impacted the entire application'. The error message is 'division by zero'. Below this, there are two boxes: 'Why did this happen?' (explaining it's a monolithic application) and 'What should you do in the lab?' (listing three steps). A red 'HTTP 500' badge is in the top right corner. Navigation buttons 'Back to Events' and 'Login' are at the bottom.

```
INFO:     127.0.0.1:51731 - "GET /checkout HTTP/1.1" 500 Internal Server Error
ERROR:    Exception in ASGI application
```

# PART-4

## SS3:

localhost:8000/checkout

Fest Monolith  
FastAPI • SQLite • Locust

**Checkout**  
This route is used to demonstrate a monolith crash + optimization.

Total Payable  
**₹ 9500**

After fixing + optimizing checkout logic, re-run Locust and compare results.

**What you should observe**

- One buggy feature can crash the entire monolith.
- Inefficient loops cause high response times under load.
- Optimization improves performance but architecture still scales as one unit.

Next Lab: Split this monolith into Microservices (Events / Registration / Checkout).

```
INFO:    127.0.0.1:57012 - "GET /checkout HTTP/1.1" 200 OK
```

# PART-5

SS4:

The screenshot shows two windows side-by-side. On the left is the Locust web interface at [localhost:8089](http://localhost:8089), displaying a "STATISTICS" tab with a table of performance metrics for a GET request to "/checkout". On the right is a terminal window titled "C:\Windows\System32\cmd.exe" showing command-line logs related to Locust's setup and execution.

**Locust Web Interface Statistics:**

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)
GET	/checkout	13	0	9	2100	2100	169.18	4
Aggregated								
		13	0	9	2100	2100	169.18	4

**Terminal Log (C:\Windows\System32\cmd.exe):**

```
C:\SEM-6\CC_Lab\PES1UG23AM189\CC_Lab2>python -m venv .venv
Error: [Errno 13] Permission denied: 'C:\\SEM-6\\CC_Lab\\PES1UG23AM189\\CC_Lab2\\.venv\\Scripts\\python.exe'

C:\SEM-6\CC_Lab\PES1UG23AM189\CC_Lab2>.\.venv\Scripts\activate
(.venv) C:\SEM-6\CC_Lab\PES1UG23AM189\CC_Lab2>locust -f locustfile.py
[2026-01-21 11:58:27,148] NEHA-SAMSUNG-PC/INFO/locust.main: Starting Locust 2.43.1
[2026-01-21 11:58:27,149] NEHA-SAMSUNG-PC/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2026-01-21 12:02:21,985] NEHA-SAMSUNG-PC/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2026-01-21 12:02:22,002] NEHA-SAMSUNG-PC/INFO/locust.runners: All users spawned: {"CheckoutUser": 1} (1 total users)

[2026-01-21 11:58:27,149] NEHA-SAMSUNG-PC/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2026-01-21 12:02:21,985] NEHA-SAMSUNG-PC/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2026-01-21 12:02:22,002] NEHA-SAMSUNG-PC/INFO/locust.runners: All users spawned: {"CheckoutUser": 1} (1 total users)
Traceback (most recent call last):
  File "C:\\SEM-6\\CC_Lab\\PES1UG23AM189\\CC_Lab2\\.venv\\Lib\\site-packages\\gevent\\_ffi\\loop.py", line 279, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused d-argument
KeyboardInterrupt
2026-01-21T06:33:28Z
[2026-01-21 12:03:28,579] NEHA-SAMSUNG-PC/INFO/locust.main: Shutting down (exit code 0)
Type           Name          # reqs      # fails
Avg           Min          Max       Med |   req/s failures/s |
-----|-----|-----|-----|-----|-----|-----|
GET           /checkout     19        0        0.00  0(0.00%) |
118          3        2087     8        0.66  0.00  0(0.00%) |
-----|-----|-----|-----|-----|-----|-----|
Aggregated    Aggregated   19        0        0.00  0(0.00%) |
118          3        2087     8        0.66  0.00  0(0.00%) |

Response time percentiles (approximated)
Type           Name          50%      66%      75%
80%          90%          95%      98%      99%      99.9%  99.99%  100% # reqs
-----|-----|-----|-----|-----|-----|-----|-----|-----|
GET           /checkout     8        9        11
12          17        2100     2100     2100     2100     2100     2100     19
-----|-----|-----|-----|-----|-----|-----|-----|-----|
Aggregated    Aggregated   8        9        11
12          17        2100     2100     2100     2100     2100     2100     19

(.venv) C:\SEM-6\CC_Lab\PES1UG23AM189\CC_Lab2>
```

## **PART-6**

**SS5:**

The screenshot shows two windows side-by-side. On the left is the Locust web interface at `localhost:8089`, displaying real-time statistics for a test. The main table shows a single `GET /checkout` request type with 19 requests, 0 fails, and a median response time of 7ms. Below this is an aggregated view. The top navigation bar has tabs for STATISTICS, CHARTS, FAILURES, EXCEPTIONS, and CURRENT RATIO. On the right is a Command Prompt window running on Windows, showing the output of a Locust test. The log shows ramping up to 1 user per second and then all users spawning. It includes a stack trace for a `KeyboardInterrupt` and detailed response time percentiles for the `/checkout` endpoint.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Mi (m)
GET	/checkout	19	0	7	2100	2100	115.69	3
Aggregated								
		19	0	7	2100	2100	115.69	3

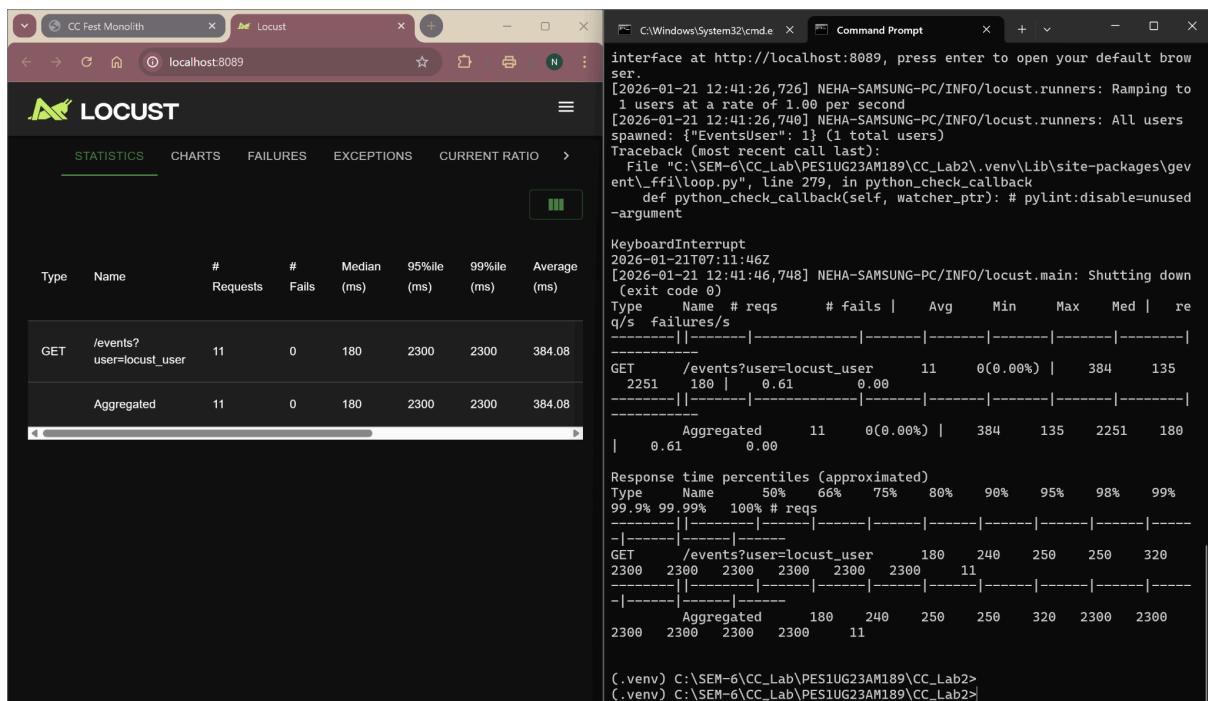
Response time percentiles (approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%
GET	/checkout	2100	2100	2100	2100	2100	2100	2100	2100
		7	8	10	12	23	2100	2100	2100
		2100	2100	2100	2100	2100	2100	2100	2100
		7	8	10	12	23	2100	2100	2100

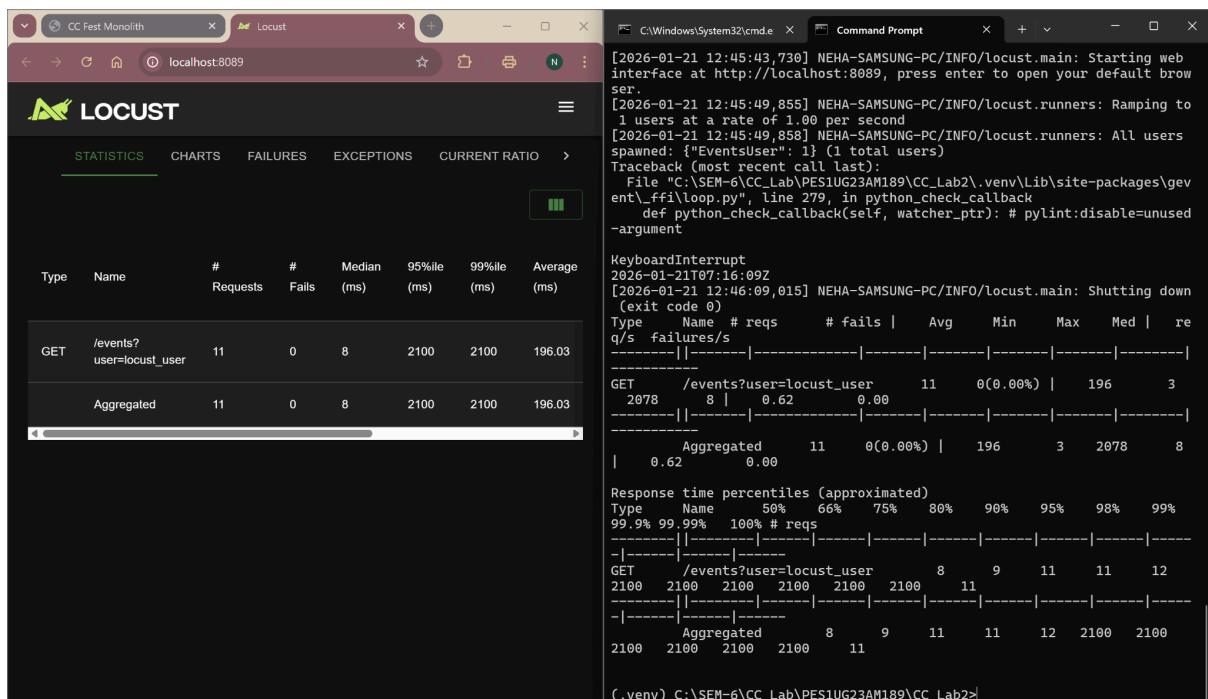
# PART-7

## Route 1: /events

### SS6 (BEFORE optimization):

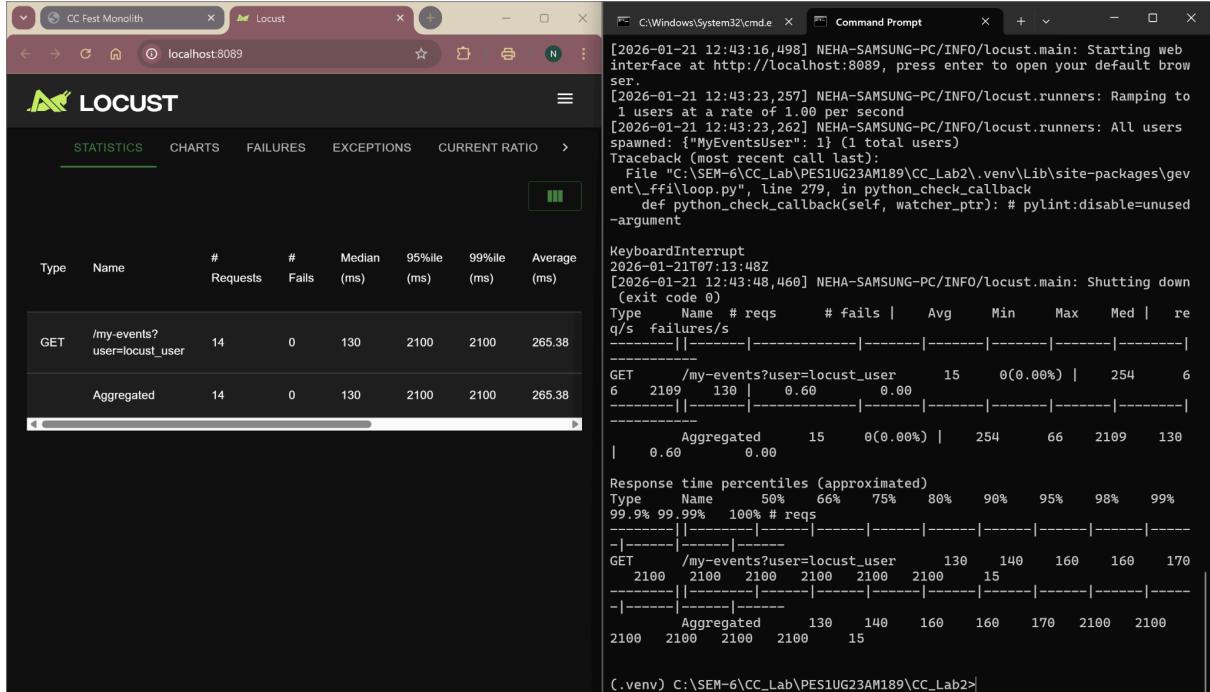


### SS7 (AFTER optimization):

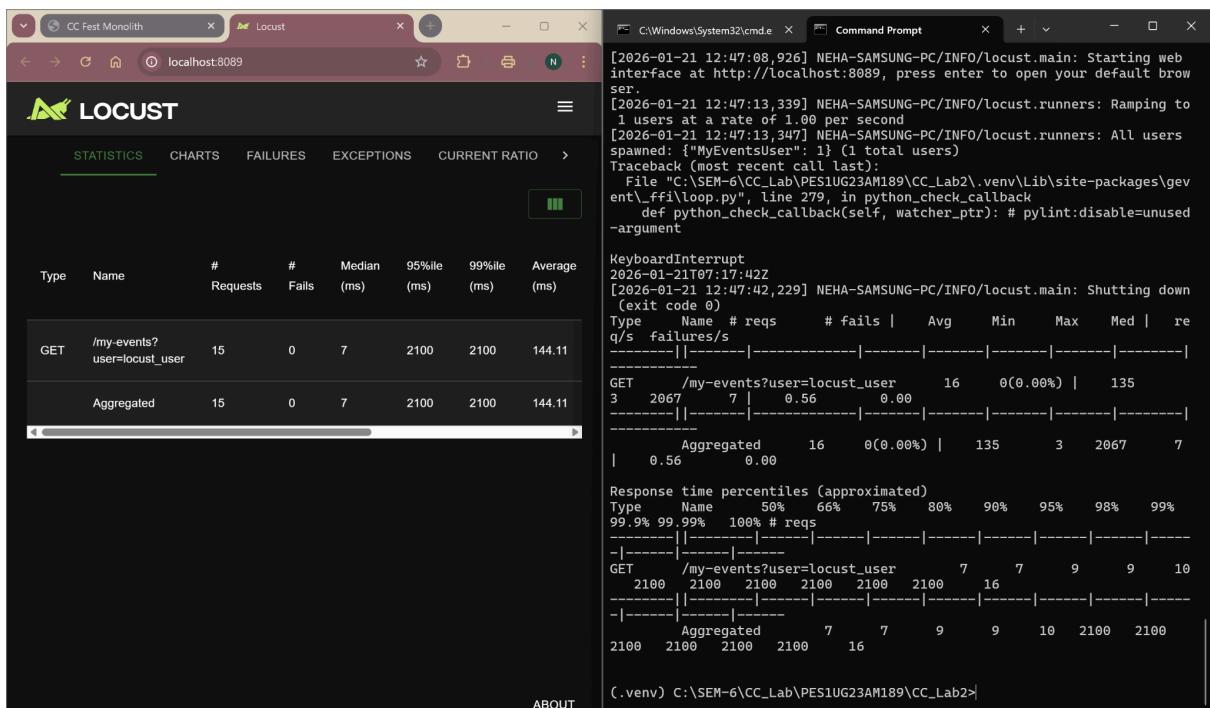


## Route 2: /my-events

### SS8 (*BEFORE optimization*):



## **SS9 (AFTER optimization):**



## Explanation of optimizations:

### Route 1: /events

- **What was the bottleneck?**

The bottleneck was unnecessary CPU-heavy computation inside the route (for i in range(3000000)), which wasted time on every request.

- **What change did you make?**

Removed/reduced the unnecessary loop (dummy computation) so the route mainly performs the database query and returns the template.

- **Why did the performance improve?**

Because the server stopped spending extra CPU time on useless calculations, the request finished faster and response time reduced.

### Route 2: /my-events

- **What was the bottleneck?**

The bottleneck was extra CPU work inside the route (for \_ in range(1500000)), which slowed down every request even after fetching the required data.

- **What change did you make?**

Removed/reduced the dummy loop and kept only the required database query + response rendering.

- **Why did the performance improve?**

Because the endpoint no longer wastes CPU cycles, it processes requests quicker, so the average response time improved under Locust load.

---