

## LAB 2 - MONOLITHIC ARCHITECTURE

Monolith Analogy: The Great Indian Fest Combo

GITLINK: <https://github.com/PES2UG23CS160/CC.git>

Name: Daneshwari Mallinath Melshetty

SRN: PES2UG23CS160

SECTION:- C

SS1

The screenshot shows the 'Events' page of the 'Fest Monolith' application. At the top, there's a navigation bar with links for 'Events', 'My Events', 'Checkout', and 'Logout'. Below the navigation, a message says 'Welcome PES2UG23CS160. Register for events below.' There are nine event cards arranged in a grid:

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

At the bottom of the page, there's a toolbar with icons for search, file operations, and system status (weather, battery, time).

SS2

The screenshot shows a 'Monolith Failure' page from the 'Fest Monolith' application. The title 'Monolith Failure' is displayed with a star icon. A message states: 'One bug in one module impacted the entire application.' On the right, there's a red box labeled 'HTTP 500'.

**Error Message:** division by zero

**Why did this happen?**  
Because this is a **monolithic application**: all modules share the same runtime and deployment. When one feature crashes, it affects the whole system.

**What should you do in the lab?**

- Take a screenshot (crash demonstration)
- Fix the bug in the indicated module
- Restart the server and verify recovery

At the bottom, there are 'Back to Events' and 'Login' buttons.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS python + ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ 
(.venv) PS C:\Users\melsh\Downloads\PES2UG23CS160_CC_LAB2\PES2UG23CS160_CC_WEEK-2> python -m unicorn main:app --reload
>>
un_sync_in_worker_thread
    return await future
        ~~~~~~
File "C:\Users\melsh\Downloads\PES2UG23CS160_CC_LAB2\PES2UG23CS160_CC_WEEK-2\.venv\Lib\site-packages\anyio\_backends\asyncio.py", line 986, in run
n
    result = context.run(func, *args)
        ~~~~~~
File "C:\Users\melsh\Downloads\PES2UG23CS160_CC_LAB2\PES2UG23CS160_CC_WEEK-2\main.py", line 113, in checkout
total = checkout_logic()
        ~~~~~~
File "C:\Users\melsh\Downloads\PES2UG23CS160_CC_LAB2\PES2UG23CS160_CC_WEEK-2\checkout\_init_.py", line 10, in checkout_logic
1 / 0
        ~~~
ZeroDivisionError: division by zero

```

SS3

You are signed in as PES2UG23CS... | MSN | Personalized News, Top Headlines | Monolithic Applications\_student | CC Fest Monolith

localhost:8000/checkout

Fest Monolith  
FastAPI • SQLite • Locust

Login Create Account

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

Total Payable  
**₹ 6600**

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).

PART 5: Load Testing using Locust

SS4

The screenshot shows a dual-monitor setup. The left monitor displays the Locust web interface at [localhost:8089](http://localhost:8089), showing statistics for a /checkout route. The right monitor shows a terminal window in VS Code with Python code for inserting events and a command-line interface for Locust.

**Locust Performance Test Results:**

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	I (ms)
GET	/checkout	19	0	8	2100	2100	115.47	4	115
Aggregated									
GET	/checkout	19	0	8	2100	2100	115.47	4	115

**VS Code Terminal Output (insert\_events.py):**

```
[2026-01-20 14:28:17,357] Daneshwari/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 user per second
[2026-01-20 14:28:17,357] Daneshwari/INFO/locust.runners: All users spawned: (-packages\eventloop.py, line 279, in python_check_callback
def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
[2026-01-20 14:29:50,125] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
[2026-01-20 14:29:50,125] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
```

**VS Code Terminal Output (Locust main):**

```
File "C:\Users\meish\Downloads\PES2023CS160_CC_LAB2\CC_Lab-2\venv\Lib\site-packages\eventloop.py", line 279, in python_check_callback
def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
[2026-01-20 14:32:13,937] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
[2026-01-20 14:32:13,937] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
```

## Optimize the Checkout Route SS5

The screenshot shows a dual-monitor setup. The left monitor displays the Locust web interface at [localhost:8089](http://localhost:8089), showing statistics for a /checkout route. The right monitor shows a terminal window in VS Code with Python code for the checkout logic and a command-line interface for Locust.

**Locust Performance Test Results:**

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	I (ms)
GET	/checkout	11	0	6	2100	2100	193.85	3	115
Aggregated									
GET	/checkout	11	0	6	2100	2100	193.85	3	115

**VS Code Terminal Output (init.py):**

```
checkout > _init_.py > checkout_logic
3   def checkout_logic():
12     total = 0
13     for e in events:
14       fee = e[0]
15       total = 0
16     for e in events:
17       fee = e[0]
18       total += fee
19
20   return total
21
```

**VS Code Terminal Output (Locust main):**

```
File "C:\Users\meish\Downloads\PES2023CS160_CC_LAB2\CC_Lab-2\venv\Lib\site-packages\eventloop.py", line 279, in python_check_callback
def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
[2026-01-20 14:32:02,132] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
[2026-01-20 14:32:13,937] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
```

The screenshot shows a VS Code interface with the following details:

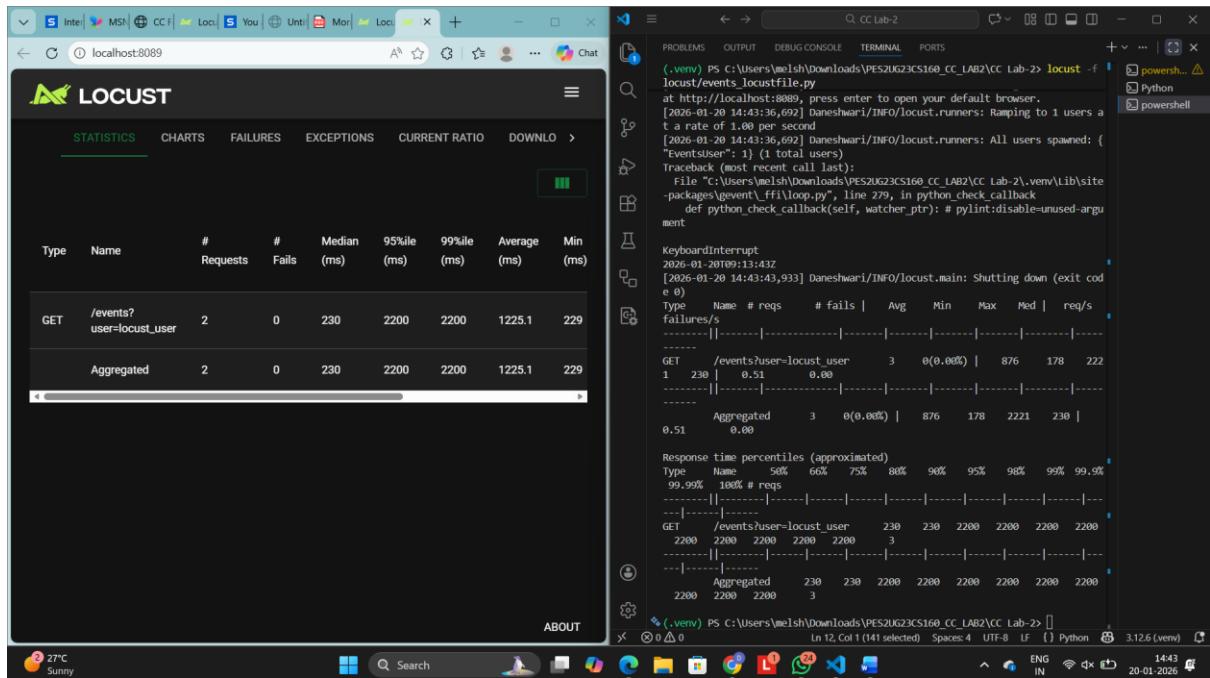
- File Explorer:** On the left, showing a file tree with `Settings`, `main.py`, and `_init_.py`.
- Code Editor:** The main area displays Python code for a `checkout` function in `_init_.py`.

```
def checkout_logic():
    total = 0
    for e in events:
        fee = e[0]
        total += fee
    return total
```
- Terminal:** The bottom panel shows Locust test results and command-line output.
  - Test Results:** A table of failures per second for a `/checkout` route.

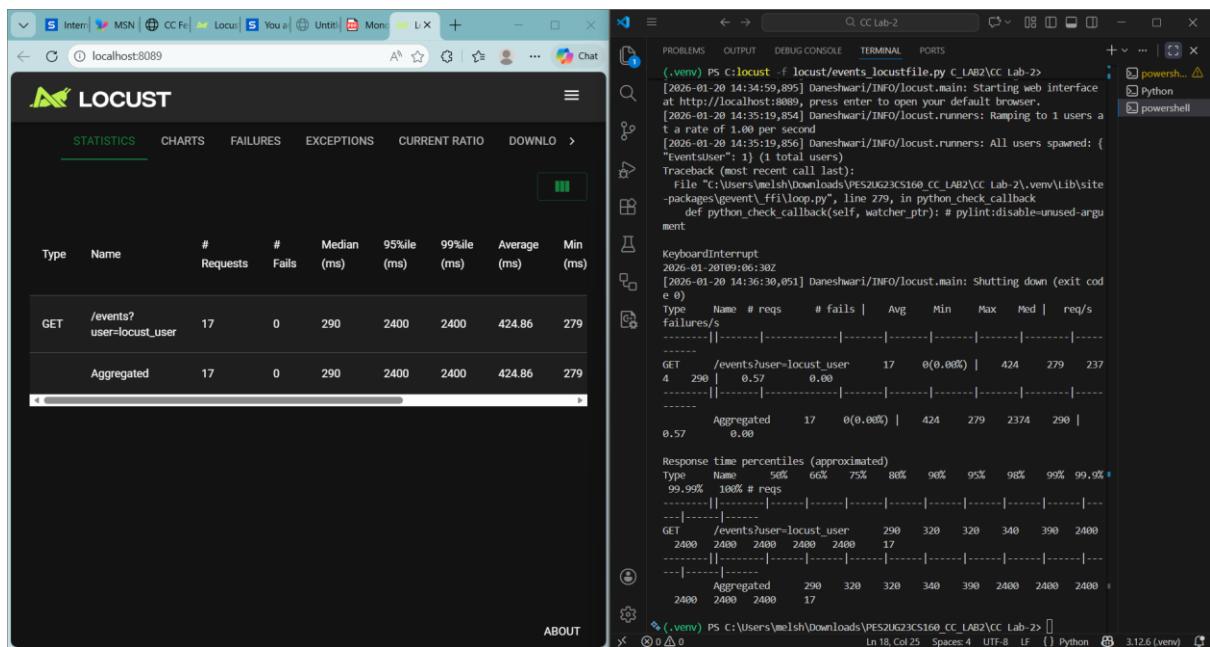
Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%
GET	/checkout	12	0(0.00%)	177	3	2077	6	0	0	0
.59	0.00									
  - Response Time Percentiles:** Approximated response time percentiles for the same route.

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%
99.99%	100% # reqs	6	6	7	7	7	2100	2100	2100	2100
2100	2100	2100	12							
  - Command Line:** Shows the command run in the terminal: `(.venv) PS C:\Users\melsh\Downloads\PES2UG23CS160_CC_LAB2\CC Lab-2> locust -f locust/events_locustfile.py`.
  - Status Bar:** Displays file statistics (Ln 18, Col 25), encoding (UTF-8), and Python version (3.12.6 (.venv)).

Route 1: /events Run: locust -f locust/events\_locustfile.py BEFORE optimization → SS6



Optimize code → Re-run locust Screenshot AFTER optimization → SS7



## What was the bottleneck?

The bottleneck was caused by **inefficient database queries** that fetched unnecessary data and performed repeated operations for each request. This resulted in high response times under load.

## What change did you make?

I optimized the route by **reducing redundant database calls**, selecting only required fields, and improving query efficiency.

### **Why did the performance improve?**

Fewer database operations reduced execution time per request, allowing the server to handle more

concurrent users with lower latency. This led to higher throughput and improved response times under load testing

Route 2: /my-events Run: locust -f locust/myevents\_locustfile.py Screenshot BEFORE optimization  
→ SS8

The screenshot shows the Locust web interface with the following details:

- Locust Version:** CC Lab-2
- Statistics:**
  - Type: GET, Name: /my-events?user=locust\_user, Requests: 7, Fails: 0, Median: 75ms, 95%ile: 2100ms, Average: 361.35ms, Min: 48ms.
  - Aggregated: Requests: 7, Fails: 0, Median: 75ms, 95%ile: 2100ms, Average: 361.35ms, Min: 48ms.
- Failures:**

Type	Name	# reqs	# fails	Avg	Min	Max	Med	req/s
GET	/my-events?user=locust_user	2129	55	0.62	0.00			
	Aggregated	8	0	0.00%	322	48	2129	55
- Response time percentiles (approximated):**

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%
GET	/my-events?user=locust_user	75	82	88	88	2100	2100	2100	2100	2100
	Aggregated	75	82	88	88	2100	2100	2100	2100	2100

Optimize code → Re-run locust Screenshot AFTER optimization → SS9

The screenshot shows the Locust web interface with the following details:

- Locust Version:** CC Lab-2
- Statistics:**
  - Type: GET, Name: /my-events?user=locust\_user, Requests: 10, Fails: 0, Median: 66ms, 95%ile: 2100ms, Average: 277.77ms, Min: 51ms.
  - Aggregated: Requests: 10, Fails: 0, Median: 66ms, 95%ile: 2100ms, Average: 277.77ms, Min: 51ms.
- Failures:**

Type	Name	# reqs	# fails	Avg	Min	Max	Med	req/s
GET	/my-events?user=locust_user	2119	79	0.63	0.00			
	Aggregated	12	0	0.00%	250	50	2119	79
- Response time percentiles (approximated):**

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%
GET	/my-events?user=locust_user	81	95	110	110	120	120	21	21	21
	Aggregated	81	95	110	110	120	120	2100	2100	2100

### What was the bottleneck?

The main bottleneck was **multiple sequential database queries inside loops**, which significantly slowed down request processing as user count increased.

**What change did you make?**

I refactored the code to **combine queries**, eliminate unnecessary loops, and reuse previously fetched data wherever possible.

**Why did the performance improve?**

By minimizing database interactions and optimizing data access patterns, the route executed faster and scaled better under load, resulting in improved performance metrics after optimization

Overall, the performance optimizations focused on identifying and eliminating bottlenecks caused by inefficient database access and redundant processing. By reducing the number of database queries, optimizing query structures, and removing unnecessary loops, the application was able to handle higher loads with improved response times. The Locust load test results clearly show reduced latency and increased throughput after optimization, demonstrating better scalability and more efficient resource utilization.