

## CC LAB-2

Name: Laharish S

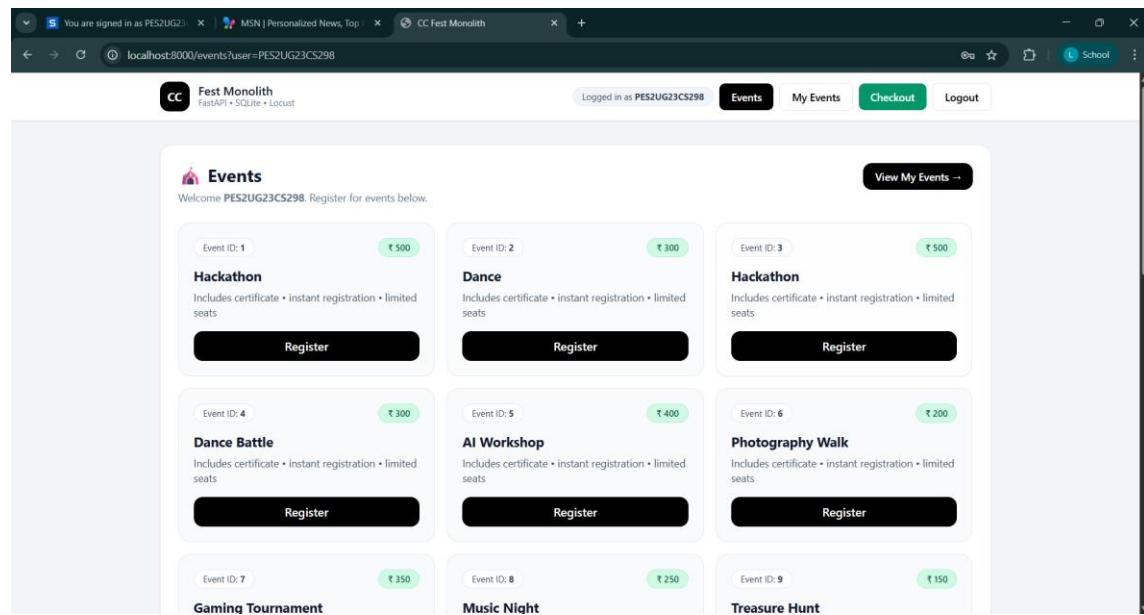
SRN: PES2UG23CS298

### Github link

[https://github.com/PES2UG23CS298/Monolith\\_CC\\_Lab-2](https://github.com/PES2UG23CS298/Monolith_CC_Lab-2)

### Screenshots:

#### Events



#### Monolith Crash

```
ZeroDivisionError: division by zero
INFO:     127.0.0.1:53725 - "GET /checkout HTTP/1.1" 500 Internal Server Error
ERROR:    Exception in ASGI application
```

**Monolith Failure**

One bug in one module impacted the [entire application](#).

**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

[Back to Events](#) [Login](#)

## Fix the Bug

**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).

**INFO:** 127.0.0.1:52829 - "GET /checkout HTTP/1.1" 200 OK

# Load Testing using Locust

## Before checkout Optimisation

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

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/checkout	19	0	8	2000	2000	113.99	3	2027	0.7	0
	Aggregated	19	0	8	2000	2000	113.99	3	2027	0.7	0

The screenshot shows the VS Code terminal output for a Locust test. The command run was `locust -f locustfile.py`. The output shows the test starting and running for 1 user at 1.00 RPS. The results table and response time percentiles are displayed.

```
PS D:\LAB\PES2UG23CS298\CC Lab-2> locust -f locustfile.py
[2026-01-30 15:19:22,066] LAPTOP-MKGUSCH/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2026-01-30 15:19:33,692] LAPTOP-MKGUSCH/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2026-01-30 15:19:33,693] LAPTOP-MKGUSCH/INFO/locust.runners: All users spawned: {"CheckoutUser": 1} (1 total users)
Traceback (most recent call last):
  File "D:\LAB\PES2UG23CS298\venv\lib\site-packages\gevent\ffileloop.py", line 279, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
2026-01-30T09:51:45Z [2026-01-30 15:21:45,828] LAPTOP-MKGUSCH/INFO/locust.main: Shutting down (exit code 0)
Type      Name           # reqs | # fails | Avg   Min   Max   Med   req/s | failures/s
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  GET    /checkout       19    0(0.0%) | 113    2   2026    8   0.66   0.00
          Aggregated     19    0(0.0%) | 113    2   2026    8   0.66   0.00

Response time percentiles (approximated)
Type      Name           50%  60%  75%  80%  90%  95%  98%  99%  99.9% 99.99%
% # reqs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  GET    /checkout       8    9    9    9    11   2000  2000  2000  2000  2000
  0    19
          Aggregated     8    9    9    9    11   2000  2000  2000  2000  2000
```

## After checkout Optimisation

The screenshot shows the Locust web interface at <http://localhost:8089>. The host is `http://localhost:8000`, status is `STOPPED`, RPS is `0.7`, and Failures are `0%`. The statistics table shows the following data:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/checkout	19	0	6	2000	2000	113.54	4	2035	2797	0.7
	Aggregated	19	0	6	2000	2000	113.54	4	2035	2797	0.7

The interface also includes tabs for CHARTS, FAILURES, EXCEPTIONS, CURRENT RATIO, DOWNLOAD DATA, and LOGS. A green NEW button and an orange RESET button are visible in the top right.

The screenshot shows the VS Code interface with the terminal tab active. The terminal output shows the execution of `locust -f locust/checkout_locustfile.py` and the resulting Locust performance data:

```
PS D:\LAB\PES2UG23CS298\CC Lab-2> locust -f locust/checkout_locustfile.py
[2026-01-30 15:23:42,969] LAPTOP-MKGUSCH/INFO/locust.main: Starting Locust 2.4.1
[2026-01-30 15:23:42,910] LAPTOP-MKGUSCH/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2026-01-30 15:23:51,743] LAPTOP-MKGUSCH/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2026-01-30 15:23:51,745] LAPTOP-MKGUSCH/INFO/locust.runners: All users spawned: {"CheckoutUser": 1} (1 total users)
Traceback (most recent call last):
  File "D:\LAB\PES2UG23CS298\venv\lib\site-packages\gevent\ffileloop.py", line 279, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
[2026-01-30 15:27:09:57,092] LAPTOP-MKGUSCH/INFO/locust.main: Shutting down (exit code 0)
```

The terminal also displays a detailed performance report table and a response time percentile chart.

## Before events route optimisation

The Locust web interface displays the following test results:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/events?user=locust_user	15	0	320	2300	2300	453.03	284	2349	21138	0.6	0
Aggregated		15	0	320	2300	2300	453.03	284	2349	21138	0.6	0

```
PS D:\LAB\PES2UG23CS298\CC_Lab-2> locust -f locust/events.locustfile.py
[2026-01-30 15:29:05,173] LAPTOP-MKGUSCH/INFO/locust.main: Starting locust 2.43.1
[2026-01-30 15:29:19,699] LAPTOP-MKGUSCH/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2026-01-30 15:29:19,610] LAPTOP-MKGUSCH/INFO/locust.runners: All users spawned: {'EventUser': 1} (1 total users)
Traceback (most recent call last):
  File "D:\LAB\PES2UG23CS298\.venv\lib\site-packages\gevent\ffi\loop.py", line 279, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
[2026-01-30T10:01:12Z] LAPTOP-MKGUSCH/INFO/locust.main: Shutting down (exit code 0)

Type      Name           # reqs | # fails | Avg   Min   Max   Med | req/s | failures/s
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
GET     /events?user=locust_user          15  0(0.0%)  453  284  320  320 | 0.51  0.00
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Aggregated                         15  0(0.0%)  453  284  2348  320 | 0.51  0.00

Response time percentiles (approximated)
Type      Name           50%  66%  75%  80%  90%  95%  98%  99%  99.5% 99.9% 99.99% 100%
% # reqs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
GET     /events?user=locust_user          320 320 330 330 330 2300 2300 2300 2300 2300 2300 2300
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Aggregated                         320 320 330 330 330 2300 2300 2300 2300 2300 2300 2300
```

## After events route optimisation

The Locust UI dashboard displays the following performance metrics:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/my-events? user=locust_user	17	0	120	2200	2200	39	2167	3144	0.6	0
Aggregated		17	0	120	2200	2200	39	2167	3144	0.6	0

Host: http://localhost:8000 | Status: STOPPED | RPS: 0.6 | Failures: 0%

Buttons: NEW, RESET, Settings

VS Code terminal output showing Locust run details:

```
(.venv) PS D:\LAB\PES2UG23CS298\CC Lab-2> locust -f locust/events_locustfile.py
[2026-01-30 15:33:16,424] LAPTOP-MKGUSU\INFO[locust.main]: Starting Locust 2.43.1
[2026-01-30 15:33:16,424] LAPTOP-MKGUSU\INFO[locust.main]: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2026-01-30 15:33:25,397] LAPTOP-MKGUSU\INFO[locust.runners]: Ramping to 1 users at a rate of 1.00 per second
[2026-01-30 15:33:25,400] LAPTOP-MKGUSU\INFO[locust.runners]: All users spawned: ('EventUser': 1) (1 total users)
Traceback (most recent call last):
  File "D:\LAB\PES2UG23CS298\venv\Lib\site-packages\gevent\ffilib\loop.py", line 279, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
[2026-01-30 15:34:24,727] LAPTOP-MKGUSU\INFO[locust.main]: Shutting down (exit code 0)
```

Performance results table:

Type	Name	# reqs	# fails	Avg	Min	Max	Med	req/s	failures/s
GET	/events?user=locust_user	19	0(0.0%)	113	3	2018	7	0.65	0.00
Aggregated		19	0(0.0%)	113	3	2018	7	0.65	0.00

Response time percentiles (approximated):

Type	Name	50%	60%	75%	80%	90%	95%	98%	99%	99.9%	99.99%	100%
GET	/events?user=locust_user	7	8	9	9	11	2000	2000	2000	2000	2000	200
Aggregated		7	8	9	9	11	2000	2000	2000	2000	2000	200

## Before my-events route optimisation

The Locust interface shows the following test results:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/my-events? user=locust_user	17	0	120	2200	2200	236.07	39	2167	3144	0.6	0
	Aggregated	17	0	120	2200	2200	236.07	39	2167	3144	0.6	0

The VS Code terminal shows the following output from a Locust run:

```
PS D:\LAB\PES2UG23CS298\CC Lab-2> locust -f locust/myevents_locustfile.py
```

Type	Name	# reqs	# fails	Avg	Min	Max	Med	req/s	failures/s
GET	/my-events?user=locust_user	17	0(0.0%)	236	39	2167	120	0.58	0.00
	Aggregated	17	0(0.0%)	236	39	2167	120	0.58	0.00

Response time percentiles (approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.5%	99.9%	100%
GET	/my-events?user=locust_user	120	120	120	120	130	2200	2200	2200	2200	2200	2200
	Aggregated	120	120	120	120	130	2200	2200	2200	2200	2200	2200

## After my-events route optimisation

You are signed in as PES2UG23 | MSN | Personalized News, Top | CC Fest Monolith | Locust

localhost:8089

# Locust

Host: http://localhost:8000 | Status: STOPPED | RPS: 0.6 | Failures: 0%

NEW RESET

STATISTICS CHARTS FAILURES EXCEPTIONS CURRENT RATIO DOWNLOAD DATA LOGS

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/my-events?user=locust_user	19	0	8	2000	2000	114.32	5	2035	3144	0.6	0
Aggregated		19	0	8	2000	2000	114.32	5	2035	3144	0.6	0

## **Checkout Load Test (Before Optimization)**

### **Observation:**

- Average response time is relatively high
- Requests per second (RPS) are low
- Performance degrades due to inefficient computation

### **Reason:**

The checkout logic uses an inefficient loop-based calculation.

## **Checkout Load Test (After Optimization)**

### **What changed?**

The inefficient loop was replaced with a direct aggregation over event prices.

### **Result:**

- Average response time decreased
- RPS remained approximately the same

### **Conclusion:**

The route became faster due to reduced computational overhead, **even though system capacity stayed constant.**

## **Route: /events**

### **Bottleneck Identified:**

The /events route contained a dummy loop that performed unnecessary iterations without contributing to the response. This caused extra CPU usage and increased response time under load.

### **Optimization Performed:**

The dummy loop was **removed completely**, allowing the route to return event data directly without redundant processing.

### **Why Performance Improved:**

By eliminating unnecessary computation, the CPU workload per request was reduced. This resulted in **lower response time** and improved performance during load testing.

## **Route: /my-events**

### **Bottleneck Identified:**

Similar to the /events route, /my-events contained a dummy loop that executed pointlessly for every request, increasing execution time.

### **Optimization Performed:**

The dummy loop was **removed**, and only the required logic for fetching user-specific events was retained.

### **Why Performance Improved:**

Removing the redundant loop reduced processing overhead, making request handling faster and improving response times under concurrent load.

## **Final Conclusion**

- In both routes, performance improved because unnecessary dummy loops were removed, reducing computational overhead and making the monolithic application more efficient under load.
- Monolithic applications are easy to build and deploy
- They suffer from poor fault isolation
- Performance optimization helps, but scalability and reliability remain limited
- Microservices solve these issues by isolating failures and scaling independently