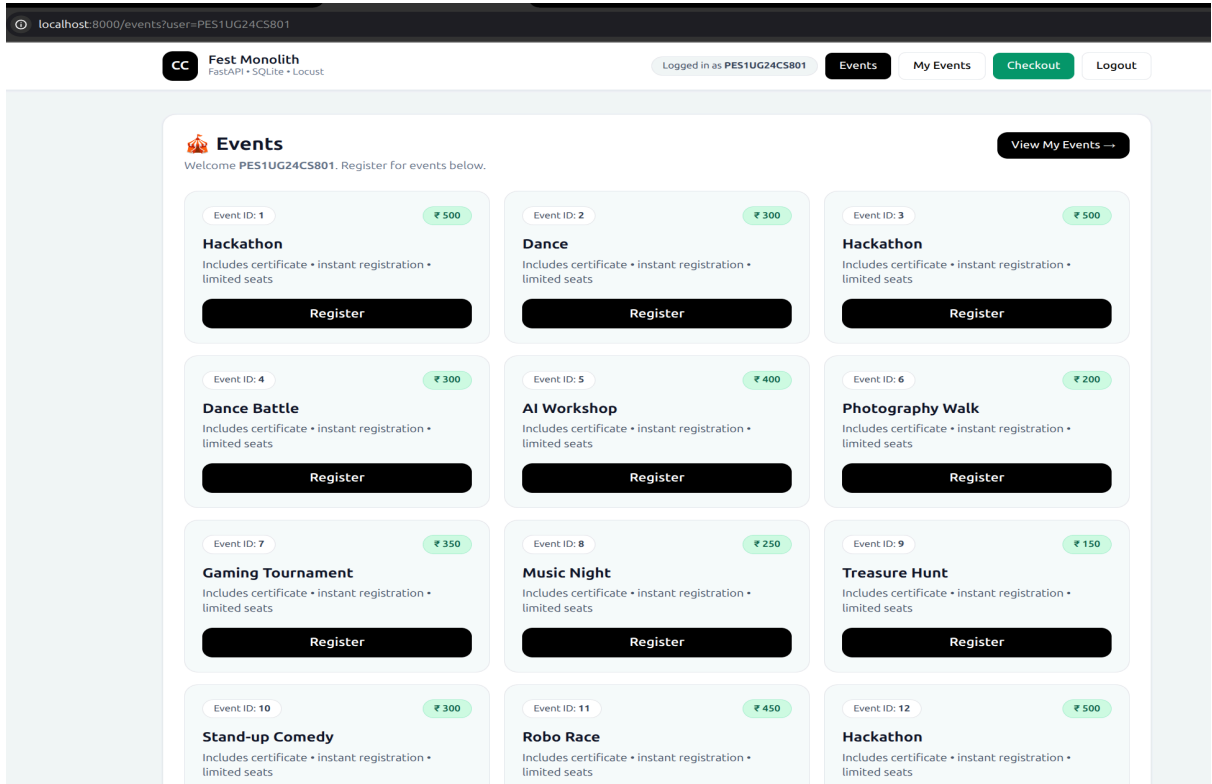


LAB 2 - MONOLITHIC ARCHITECTURE

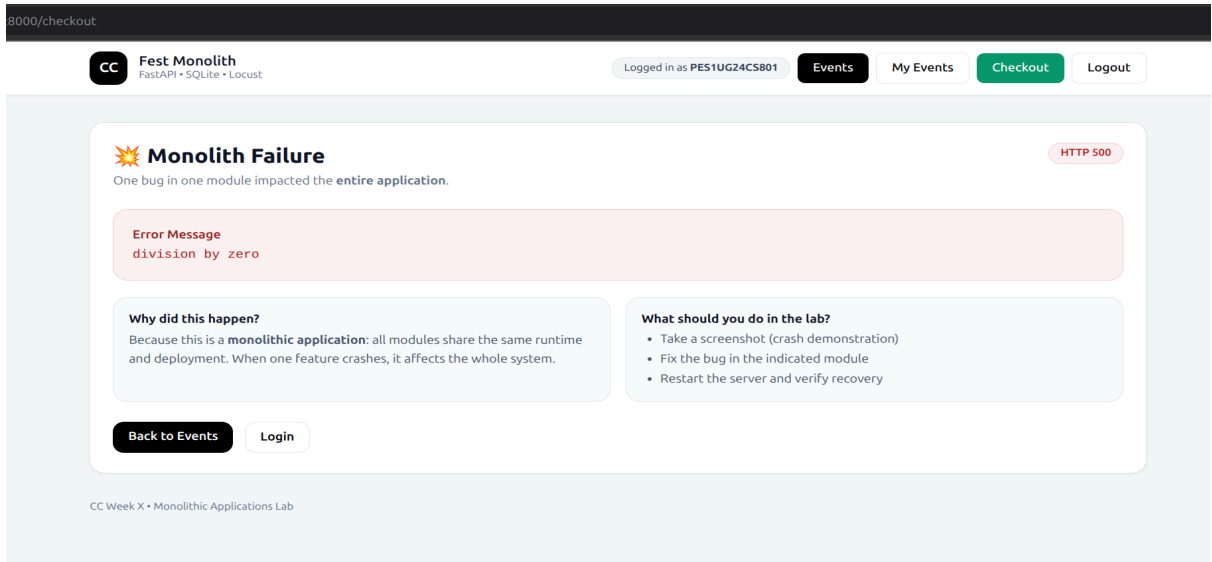
SRN: PES1UG24CS801

NAME: ABHISHEK M HIREMATH

SS1



SS2



```
INFO: 127.0.0.1:43716 - "GET /events?user=PES1UG24CS801 HTTP/1.1" 200 OK
INFO: 127.0.0.1:50172 - "GET /checkout HTTP/1.1" 500 Internal Server Error
ERROR: Exception in ASGI application
Traceback (most recent call last):
```

SS3

8000/checkout

CC

Fest Monolith

FastAPI • SQLite • Locust


Logged in as PES1UG24CS801

Events

My Events

Checkout

Logout



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).

CC Week X • Monolithic Applications Lab

INFO: 127.0.0.1:52218 - "GET /my-events?user=PES1UG24CS801 HTTP/1.1" 200 OK

INFO: 127.0.0.1:52218 - "GET /checkout HTTP/1.1" 200 OK

SS4

main.py 1

__init__.py

CC Lab-2 > checkout > __init__.py > checkout_logic

1 from database import get_db

2

3 def checkout_logic():

KeyboardInterrupt

2026-01-29T09:13:50Z

[2026-01-29 14:43:50,073] PES1UG24CS801/INFO/locust.main: Shutting down (exit code 0)

Type	Name	# reqs	# fails	Avg	Min	Max	Med	req/s	failures/s
GET	/checkout	20	0(0.00%)	3	2	5	3	0.67	0.00
Aggregated		20	0(0.00%)	3	2	5	3	0.67	0.00

Response time percentiles (approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%
GET	/checkout	3	3	4	4	4	6	6	6	6

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)	Cur RPS
GET	/checkout	20	0	3	6	6	3.36	2	6	2797	0.6
Aggregated		20	0	3	6	6	3.36	2	6	2797	0.6

SS5

VS Code Editor (main.py):

```
def checkout_logic():  
    # ...  
    # ...  
    # ...
```

Locust Interface:

- Host: http://localhost:8000
- Status: STOPPED
- RPS: 0.6
- Failures: 0%
- Buttons: NEW, RESET

Type	Name	# req/s	# fails	Avg (ms)	Min (ms)	Max (ms)	Med (ms)
GET	/checkout	20	0 (0.00%)	2	1	4	3
Aggregated							
		20	0 (0.00%)	2	1	4	

Response time percentiles (approximated):

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%
GET	/checkout	4	4	4	4	4	4	4	4
Aggregated									
		4	4	4	4	4	4	4	4

Route 1: /events

SS6

VS Code Editor (events_locustfile.py):

```
from locust import HttpUser, task, between  
# ...  
# ...
```

Locust Interface:

- Host: http://localhost:8000
- Status: RUNNING
- Users: 1
- RPS: 0.67
- Failures: 0%
- Buttons: EDIT, STOP

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)
GET	/events?user=locust_user	6	0	92	120	120	98.68	91	116	21138
Aggregated										
		6	0	92	120	120	98.68	91	116	21138

SS7

VS Code Editor (main.py):

```
@app.get("/events", response_class=HTMLResponse)  
def events(request: Request, user: str):  
    db = get_db()  
    rows = db.execute("SELECT * FROM events").fetchall()  
    # ...  
    # ...
```

Locust Interface:

- Host: http://localhost:8000
- Status: RUNNING
- Users: 1
- RPS: 0.7
- Failures: 0%
- Buttons: EDIT, STOP

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)
GET	/events?user=locust_user	11	0	3.09	15	15	4.73	3	15	21138
Aggregated										
		11	0	3.09	15	15	4.73	3	15	21138

Bottleneck:

The /events route contained an unnecessary CPU-intensive loop that executed millions of iterations on every request, even though it did not contribute to the response.

Change made:

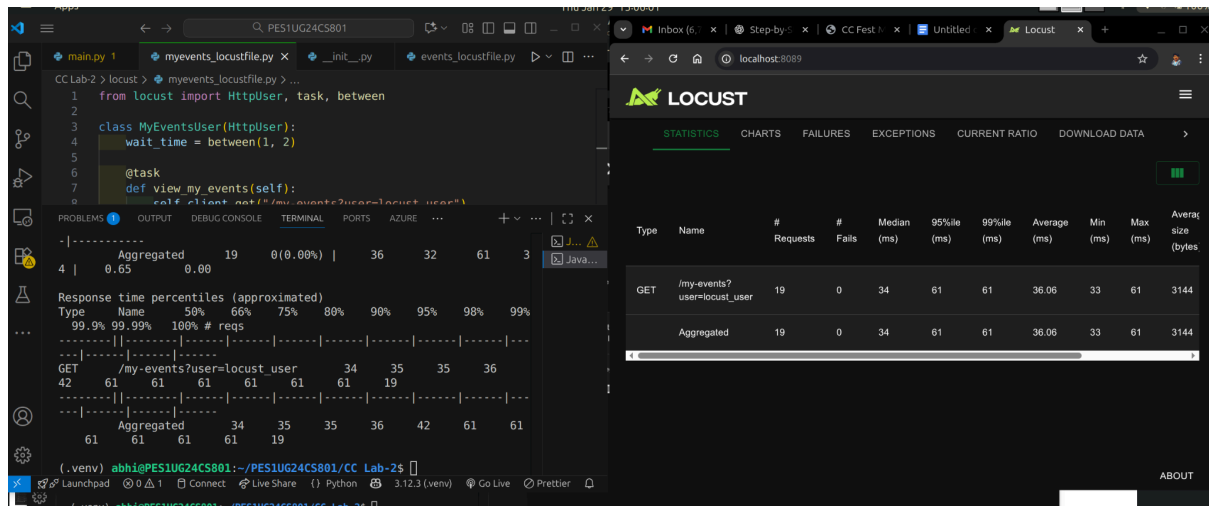
Removed the redundant computation loop and retained only the database query and template rendering logic.

Why performance improved:

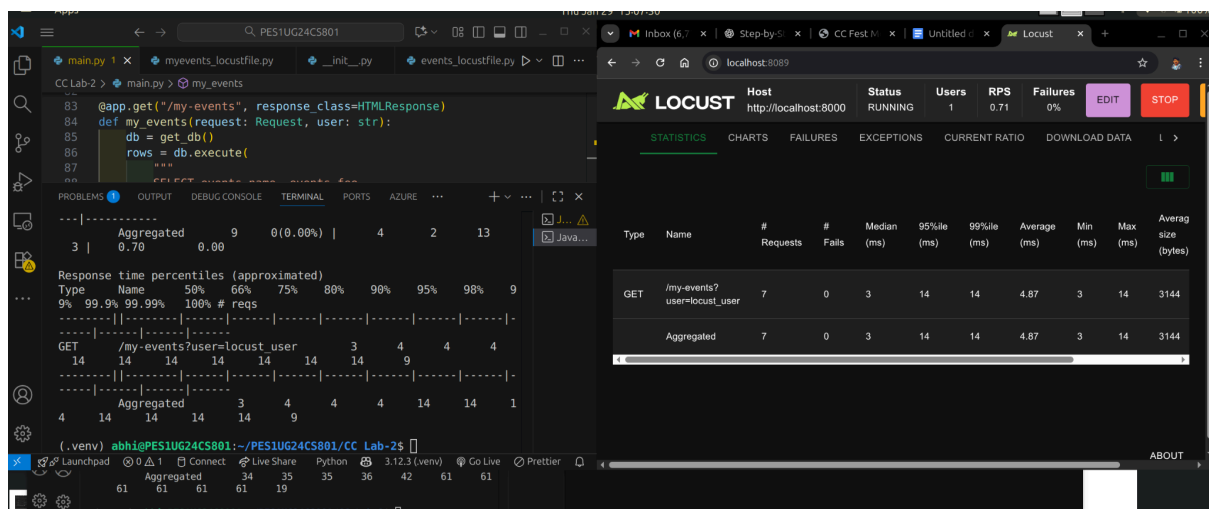
Eliminating unnecessary CPU work reduced processing time per request, leading to lower response time and improved throughput.

Route 2: /my-events

SS8



SS9



Bottleneck:

The /my-events route had an artificial delay caused by a large loop that performed meaningless operations for every incoming request.

Change made:

Removed the dummy loop and allowed the route to execute only the required database join query and response rendering.

Why performance improved:

By removing the extra computation, the server spent less CPU time per request, resulting in faster responses and better overall performance.

GitHub:-

<https://github.com/Abhi-DevHub/CC-Lab-2-MONOLITHIC-ARCHITECTURE.git>