

CC-Semester-6

LAB-2

Date: 20/01/2026

Name: A Shri Karthik

SRN: PES2UG23CS003

Section: A

SS1

The screenshot shows the 'Events' page of the Fest Monolith application. At the top, there is a header bar with the Fest Monolith logo, a sign-in link, and navigation buttons for 'Events', 'My Events', 'Checkout', and 'Logout'. Below the header, there is a section titled 'Events' with a sub-section 'View My Events'. The main content area displays nine event cards arranged in a grid:

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

SS2

The screenshot shows a 'Monolith Failure' error page from the Fest Monolith application. At the top, there is a header bar with the Fest Monolith logo, a sign-in link, and navigation buttons for 'Events', 'My Events', 'Checkout', and 'Logout'. The main content area features a large red banner with the text 'HTTP 500' and the heading 'Monolith Failure'. Below the banner, there is a message stating 'One bug in one module impacted the entire application.' and a red box containing the 'Error Message' 'division by zero'. To the right, there is a red box containing the question 'Why did this happen?' followed by the text 'Because this is a monolithic application: all modules share the same runtime and deployment. When one feature crashes, it affects the whole system.' To the left, there is another red box containing the question 'What should you do in the lab?' followed by a bulleted list:

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

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

SS3

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).

SS4

LOCUST

STATISTICS CHARTS FAILURES EXCEPTIONS CURRENT RATIO DOW >

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)
GET	/checkout	21	0	4	10	2100	101.57	3	2
Aggregated									
21 0 0.00%									

insert_events.py

```
from database import get_db
db = get_db()
# create table if not exists
db.execute('''
CREATE TABLE IF NOT EXISTS events (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    fee INTEGER
)
''')
events = [
    ("Hackathon", 500),
    ("Dance Battle", 300),
    ("AI Workshop", 400),
    ("Photography Walk", 200),
    ("Gaming Tournament", 350),
]
```

Response time percentiles (approximated)

Type	Name	96%	95%	98%	99%	99.9%	99.99%	100%	# reqs	50%	66%	75%	80%
GET	/checkout	5	5	10	2100	2100	2100	2100	21	4	4	5	
Aggregated													
5 5 10 2100 2100 2100 2100 2100 21 4 4 5													

(venv) PS C:\Users\Shri Karthik\Desktop\SEM-6-STUFF\CC\PES2UG23CS003\PES2UG23CS003>

SS5

The screenshot shows a browser window with two tabs open. The left tab displays Locust test results for a '/checkout' endpoint, showing metrics like # Requests, # Fails, Median, 95%ile, 99%ile, Average, Min, and Max. The right tab is a terminal session showing Python code for a '_init_.py' file, which includes a 'checkout_logic()' function. Below the terminal is a 'requirements.txt' file. A status bar at the bottom indicates '0.68 0.00'.

localhost:8089

10 Best Shoulder Ex... Get a "6 Pack" in 22... The PERFECT Shoul... Can't Get a Muscula...

LOCUST

STATISTICS CHARTS FAILURES EXCEPTIONS CURRENT RATIO DOW >

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)
GET	/checkout	20	0	3	2000	2000	105.36	2	2000
Aggregated									
		20	0	3	2000	2000	105.36	2	2000

checkout > _init_.py > checkout_logic()
3 def checkout_logic():
4 # Uncomment this line initially for the crash screenshot task
5 # 1 / 0
6
7 total = 0
8 for e in events:
9 total += e[0]
10
11 return total
12
13
14
15
16
17

problems output terminal ... powershell + x

0.68 0.00

Response time percentiles (approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%
		99%	99.9%	99.99%	100%	# reqs		
GET	/checkout	3	3	4	4	5	2000	2000
		2000	2000	2000	2000	20		

The terminal has no selection to copy

SS6

The screenshot shows a dual-pane interface. The left pane displays Locust's performance statistics for a test named 'events? user=locust_user'. The right pane shows the source code for the 'checkout' logic in a file named `__init__.py`.

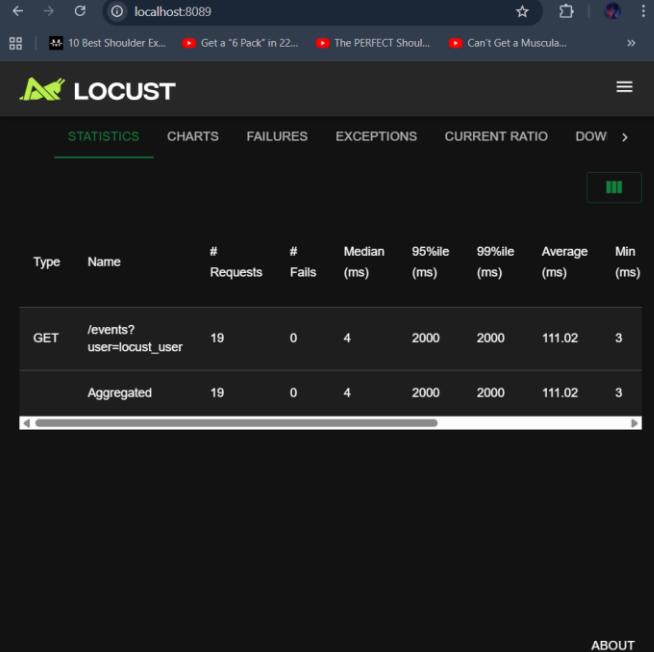
Locust Performance Statistics:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)
GET	/events? user=locust_user	18	0	79	2100	2100	193.96	74
Aggregated								
2100 2100 2100 2100 2100 18 80 83 89 89 94 2100 2100								

Code Editor (right pane):

```
checkout < __init__.py > checkout_logic
3   def checkout_logic():
5     # Uncomment this line initially for the crash screenshot task
6     # I / o
8
10    total = 0
12    for e in events:
13      total += e[0]
15
16    return total
17
```

SS7



The screenshot shows the Locust web interface at localhost:8089. The left pane displays a table of test results for a GET request to '/events?user=locust_user'. The right pane shows the source code for the application, which includes a main.py file with logic for handling events and user registration.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)
GET	/events? user=locust_user	19	0	4	2000	2000	111.02	3
Aggregated								
		19	0	4	2000	2000	111.02	3

```
File: __init__.py
def login(request: Request, username: str = Form(...), password: str = Form(...)):
    return RedirectResponse(f"/events?user={username}", status_code=302)

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

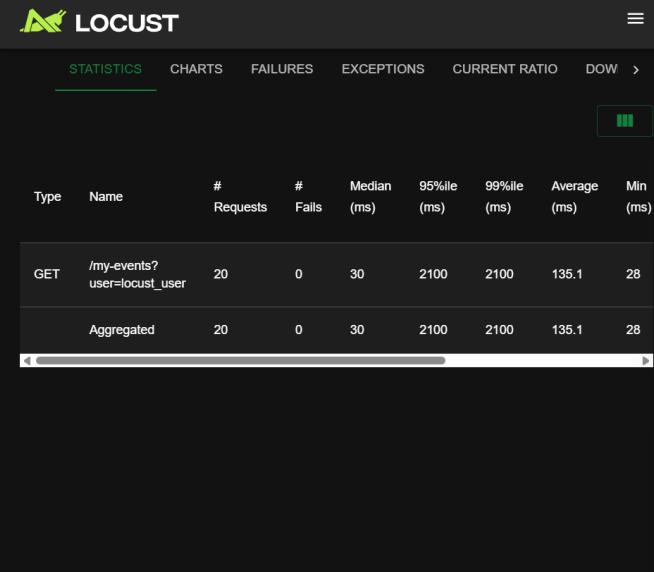
    return templates.TemplateResponse(
        "events.html",
        {"request": request, "events": rows, "user": user}
    )

@app.get("/register_event/{event_id}")
def register_event(event_id: int, user: str):
    db = get_db()
    db.execute("INSERT INTO events (id, user) VALUES (%s, %s)", (event_id, user))
    db.commit()

    return {"status": "success", "message": "Event registered successfully"}
```

```
(venv) PS C:\Users\Shri Karthik\Desktop\SEM-6-STUFF\CC\PES2UG23CS003\PE
S2UG23CS003> locust -f \locust\events_locustfile.py
[2026-01-20 15:45:00,549] KarthiksTUFF/INFO/locust.main: Starting Locus
t 2.43.1
[2026-01-20 15:45:00,550] KarthiksTUFF/INFO/locust.main: Starting web i
nterface at http://localhost:8089, press enter to open your default bro
wser.
[2026-01-20 15:45:12,658] KarthiksTUFF/INFO/locust.runners: Ramping to
1 users at a rate of 1.00 per second
[2026-01-20 15:45:12,658] KarthiksTUFF/INFO/locust.runners: All users s
pawned: {'Eventsuser': 1} (1 total users)
```

SS8



The screenshot shows the Locust web interface at localhost:8089. The left pane displays a table of test results for a GET request to '/my-events?user=locust_user'. The right pane shows the source code for the application, which includes a main.py file with logic for handling events and user registration.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)
GET	/my-events? user=locust_user	20	0	30	2100	2100	135.1	28
Aggregated								
		20	0	30	2100	2100	135.1	28

```
File: __init__.py
def login(request: Request, username: str = Form(...), password: str = Form(...)):
    return RedirectResponse(f"/events?user={username}", status_code=302)

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

    return templates.TemplateResponse(
        "events.html",
        {"request": request, "events": rows, "user": user}
    )

@app.get("/register_event/{event_id}")
def register_event(event_id: int, user: str):
    db = get_db()
    db.execute("INSERT INTO events (id, user) VALUES (%s, %s)", (event_id, user))
    db.commit()

    return {"status": "success", "message": "Event registered successfully"}
```

```
(venv) PS C:\Users\Shri Karthik\Desktop\SEM-6-STUFF\CC\PES2UG23CS003\PE
S2UG23CS003>
```

SS9

The screenshot shows a dual-pane interface. On the left is the Locust web interface displaying performance statistics for a test named 'PES2UG23CS003'. The 'STATISTICS' tab is selected, showing an aggregated request count of 20, 0 fails, and a median response time of 3ms. On the right is a code editor in PyCharm showing the Python file 'main.py'. The code contains a function 'my_events' which previously had a loop that was causing performance issues. The code has been modified to remove the loop, resulting in a significant performance improvement. A terminal window at the bottom shows the command used to run the Locust test.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)
GET	/my-events?user=locust_user	20	0	3	2000	2000	105.12
Aggregated		20	0	3	2000	2000	105.12

```
def my_events(request: Request, user: str):
    SELECT registration_id, event_id
    FROM events
    JOIN registrations ON events.id = registrations.event_id
    WHERE registrations.username=?
    """,
    (user,)
).fetchall()

return templates.TemplateResponse(
    "my_events.html",
    {"request": request, "events": rows, "user": user}
)

@app.get("/checkout", response_class=HTMLResponse)
def checkout(request: Request):
    total = checkout_logic()
    return templates.TemplateResponse(
```

2nd Case

- I) In this case, the loop was wasting time and increasing the response time for each request.
- II) We just remove the waste cycle or comment it.
- III) The loop is removed and the time to send the response for each request decreases thus improving the performance

3rd Case

- I) In this also, the loop was wasting time and increasing the response time for each request.
- II) We just remove the waste cycle or comment it.
- III) The loop is removed and the time to send the response for each request decreases thus improving the performance

Git Repo Link - <https://github.com/Kart8ik/Cloud-Computing-Lab-PES2UG23CS003>

