

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's a header bar with links for PDF, 3.9.0 Documentation, Replit, Photo Editor Pixlr Free, The Python Tutorial, c++ cheatsheet, Interactive Problems, and Titanic. Below the header, a navigation bar includes 'Events', 'My Events', 'Checkout', and 'Logout'. The main content area is titled 'Events' and displays nine event cards arranged in three rows of three. Each card contains an event ID, price, name, a brief description, and a 'Register' button.

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

SS2

The screenshot shows a 'Monolith Failure' error page. At the top, there's a header bar with links for PDF, 3.9.0 Documentation, Replit, Photo Editor Pixlr Free, The Python Tutorial, c++ cheatsheet, Interactive Problems, and Titanic 2.0 | Kaggle. Below the header, a navigation bar includes 'Events', 'My Events', 'Checkout', and 'Logout'. The main content area features a large red banner with the text 'HTTP 500' and a red star icon. Below the banner, there are three sections: 'Error Message' (division by zero), 'Why did this happen?' (explains monolithic applications), and 'What should you do in the lab?' (list of actions). At the bottom, there are 'Back to Events' and 'Login' buttons.

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).

CC Week X • Monolithic Applications Lab

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	4	10	2100	101.57	3	2

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	90%	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	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 that sums up event times.

Locust Test Results:

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

Terminal Session:

```
_init_.py
def checkout_logic():
    # Uncomment this line initially for the crash screenshot task
    # # 1 / 0

    total = 0
    for e in events:
        total += e[0]

    return total
```

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.

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								

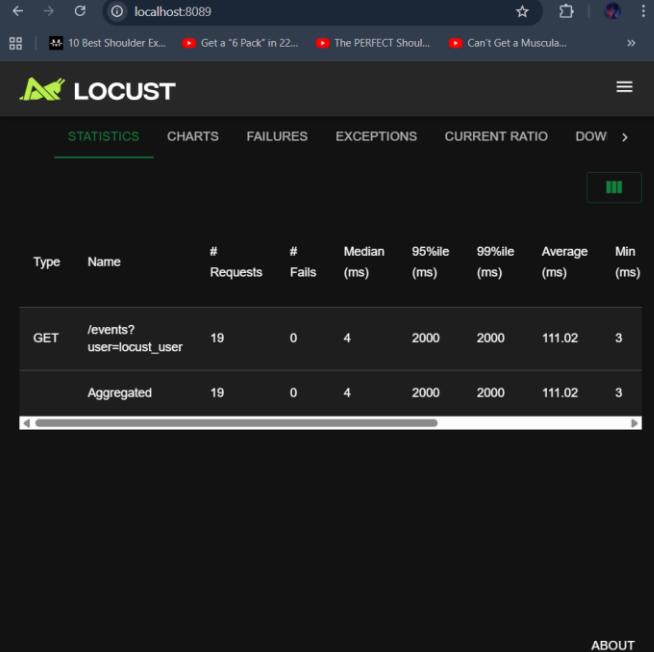
Code Editor (right pane):

```
checkout() {
    # Uncomment this line initially for the crash screenshot task
    # I / o

    total = 0
    for e in events:
        total += e[0]

    return total
}
```

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

Code snippet from main.py:

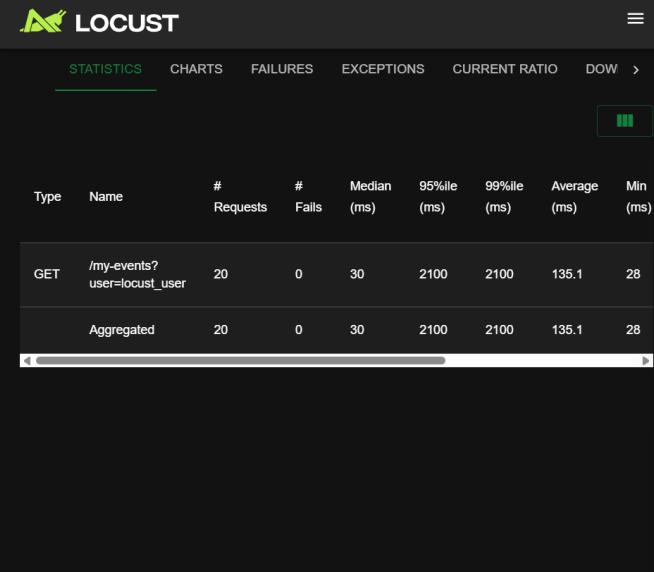
```
def login(request: Request, username: str = Form(...), password: str = Form(...)):
    return RedirectResponse(f"/events?user={username}", status_code=302)

@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()
```

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

Code snippet from main.py:

```
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()
```

SS9

The screenshot shows a dual-pane interface. On the left is the Locust web interface displaying performance statistics for a test named 'my-events?user=locust_user'. The table shows 20 requests, 0 fails, median 3ms, 95%ile 2000ms, average 105.12ms, and min 3ms. On the right is a code editor in PyCharm showing the Python file 'main.py' with the following code:

```
def my_events(request: Request, user: str):
    SELECT * 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(
        "checkout.html",
        {"total": total}
    )
```

Below the code editor is a terminal window showing command-line output. At the bottom, there are tabs for Timeline, Outline, and PES2UG23CS003.

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>