

CLOUD COMPUTING

LAB 02:

MONOLITHIC

ARCHITECTURE

Name: NIHARIKA SAHA

SRN: PES1UG23AM190

Sec: AIML-D

PART 1: Setup & Run

```
C:\COLLEGE\sem6\cc lab\actual_lab2>mkdir PES1UG23AM190
```

```
C:\COLLEGE\sem6\cc lab\actual_lab2>cd PES1UG23AM190  
C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190>cd C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190\CC Lab-2
```

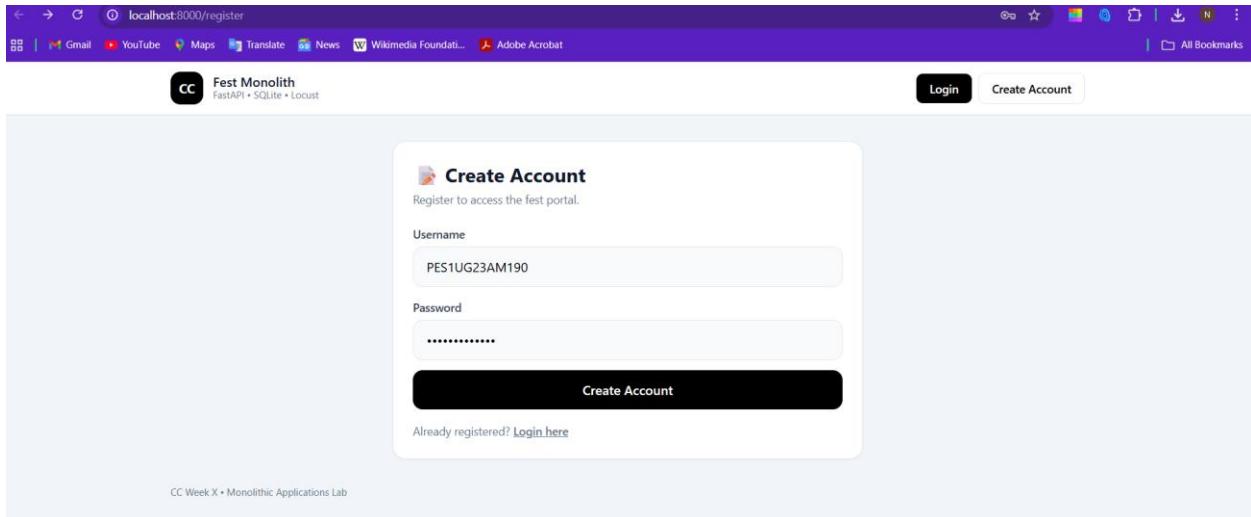
```
C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190\CC Lab-2>python -m venv .venv  
C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190\CC Lab-2>.\.venv\Scripts\activate
```

```
(.venv) C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190\CC Lab-2>pip install -r requirements.txt  
Collecting fastapi  
  Downloading fastapi-0.128.0-py3-none-any.whl (103 kB)  
    |██████████| 103 kB 2.2 MB/s  
Collecting uvicorn  
  Downloading uvicorn-0.40.0-py3-none-any.whl (68 kB)  
    |██████████| 68 kB 4.8 MB/s  
Collecting jinja2  
  Using cached jinja2-3.1.6-py3-none-any.whl (134 kB)  
Collecting python-multipart  
  Downloading python_multipart-0.0.21-py3-none-any.whl (24 kB)  
Collecting locust  
  Downloading locust-2.43.1-py3-none-any.whl (1.5 MB)  
    |██████████| 1.5 MB 6.4 MB/s  
Collecting typing-extensions>=4.8.0  
  Downloading typing_extensions-4.15.0-py3-none-any.whl (44 kB)
```

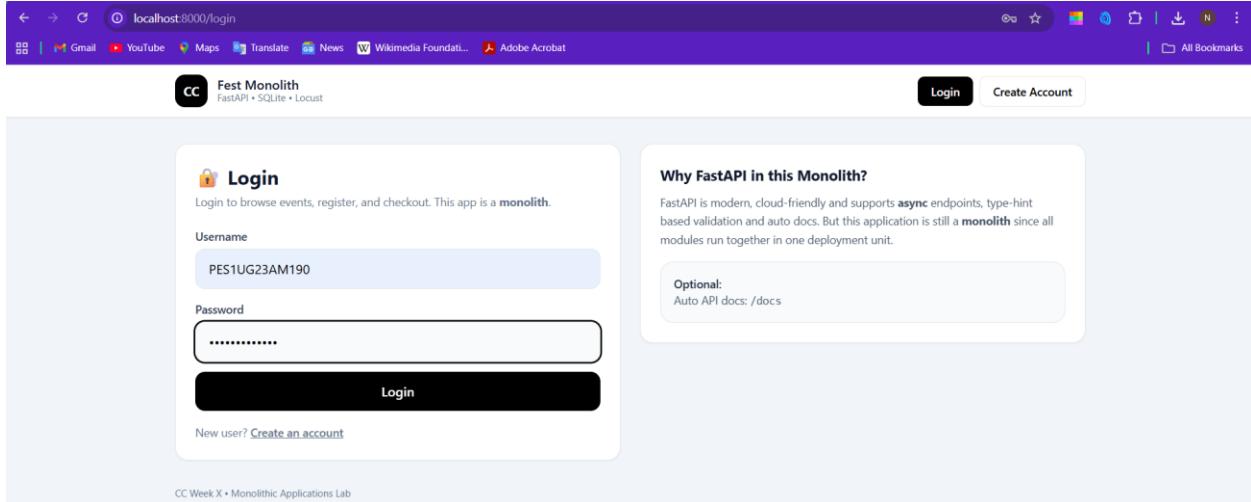
```
(.venv) C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190\CC Lab-2>python insert_events.py  
✓ Events inserted successfully!
```

```
(.venv) C:\COLLEGE\sem6\cc lab\actual_lab2\PES1UG23AM190\CC Lab-2>uvicorn main:app --reload  
INFO: Will watch for changes in these directories: ['C:\\COLLEGE\\sem6\\cc lab\\actual_lab2\\PES1UG23AM190\\CC Lab-2']  
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)  
INFO: Started reloader process [26228] using StatReload  
INFO: Started server process [16412]  
INFO: Waiting for application startup.  
INFO: Application startup complete.  
WARNING: StatReload detected changes in 'main.py'. Reloading...  
INFO: Shutting down
```

PART 2: Use the Application



The screenshot shows a web browser window with the URL `localhost:8000/register`. The title bar includes the Fast Monolith logo and the text "FastAPI + SQLite + Locust". The main content area is titled "Create Account" with a sub-instruction "Register to access the fest portal." It contains two input fields: "Username" (containing "PES1UG23AM190") and "Password" (containing "*****"). Below these is a large black "Create Account" button. At the bottom, there is a link "Already registered? [Login here](#)". The footer of the page reads "CC Week X • Monolithic Applications Lab".



The screenshot shows a web browser window with the URL `localhost:8000/login`. The title bar includes the Fast Monolith logo and the text "FastAPI + SQLite + Locust". The main content area is titled "Login" with the sub-instruction "Login to browse events, register, and checkout. This app is a **monolith**". It contains two input fields: "Username" (containing "PES1UG23AM190") and "Password" (containing "*****"). Below these is a large black "Login" button. At the bottom, there is a link "New user? [Create an account](#)". To the right of the login form, there is a sidebar with the heading "Why FastAPI in this Monolith?" and the text: "FastAPI is modern, cloud-friendly and supports **async** endpoints, type-hint based validation and auto docs. But this application is still a **monolith** since all modules run together in one deployment unit." It also includes an "Optional:" section with the text "Auto API docs: /docs". The footer of the page reads "CC Week X • Monolithic Applications Lab".

SS1: Events page loaded

The screenshot shows a web browser window with the URL `localhost:8000/events?user=PES1UG23AM190`. The page title is "Events". It displays a list of nine events, each with a "Register" button:

- Event ID: 1 - Hackathon (₹ 500)
- Event ID: 2 - Dance (₹ 300)
- Event ID: 3 - Hackathon (₹ 500)
- Event ID: 4 - Dance Battle (₹ 300)
- Event ID: 5 - AI Workshop (₹ 400)
- Event ID: 6 - Photography Walk (₹ 200)
- Event ID: 7 (partially visible) - (₹ 350)
- Event ID: 8 (partially visible) - (₹ 250)
- Event ID: 9 (partially visible) - (₹ 150)

PART 3: Observe Monolithic Failure (Crash)

SS2: Go to Checkout

The screenshot shows a web browser window with the URL `localhost:8000/checkout`. The page title is "Checkout". It displays an error message:

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

Buttons: Back to Events, Login

```
INFO:    127.0.0.1:8001 - "GET /events/user/PES1UG23AM190 HTTP/1.1" 200 OK
INFO:    127.0.0.1:58476 - "GET /checkout HTTP/1.1" 500 Internal Server Error
ERROR:   Exception in ASGI application
```

Event	Fee (₹)
Hackathon Confirmed - Fest pass	₹ 500
Dance Confirmed - Fest pass	₹ 300
AI Workshop Confirmed - Fest pass	₹ 400
Photography Walk Confirmed - Fest pass	₹ 200
Robo Race Confirmed - Fest pass	₹ 450

Proceed to Checkout →

PART 4: Fix the Bug

SS3: Fixing bug

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

Total Payable
₹ 9500

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.

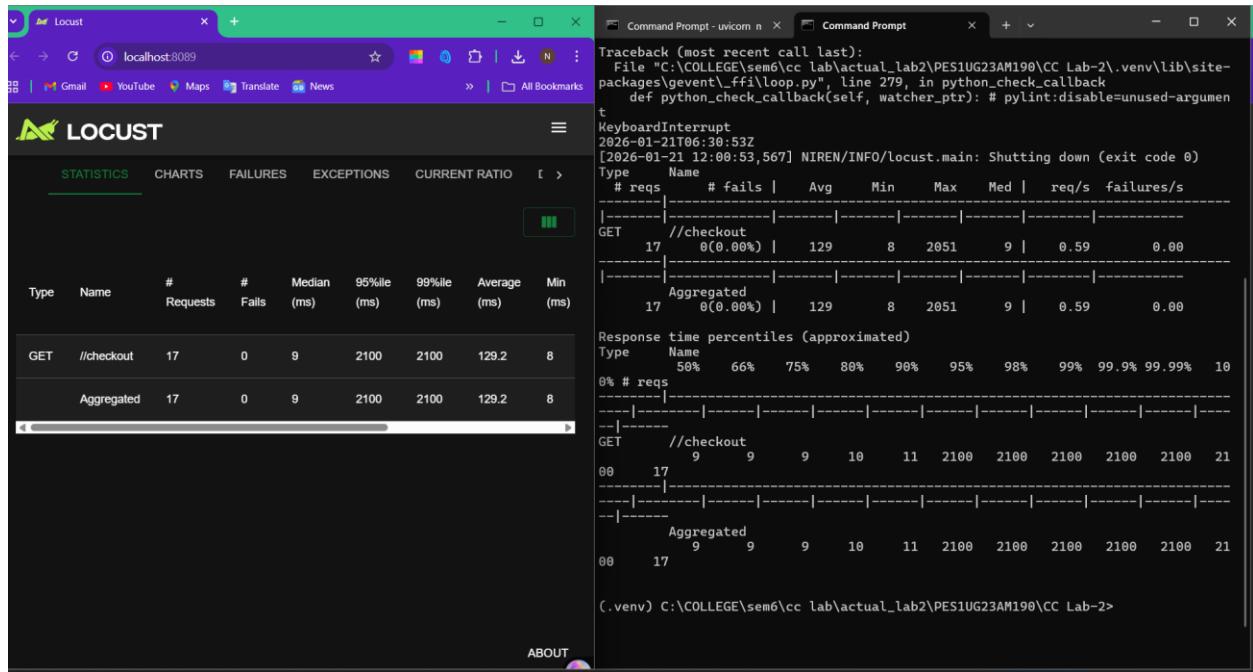
Next Lab: Split this monolith into Microservices (Events / Registration / Checkout).

```
INFO:      Application startup complete.
INFO:      127.0.0.1:61609 - "GET /checkout HTTP/1.1" 200 OK
```

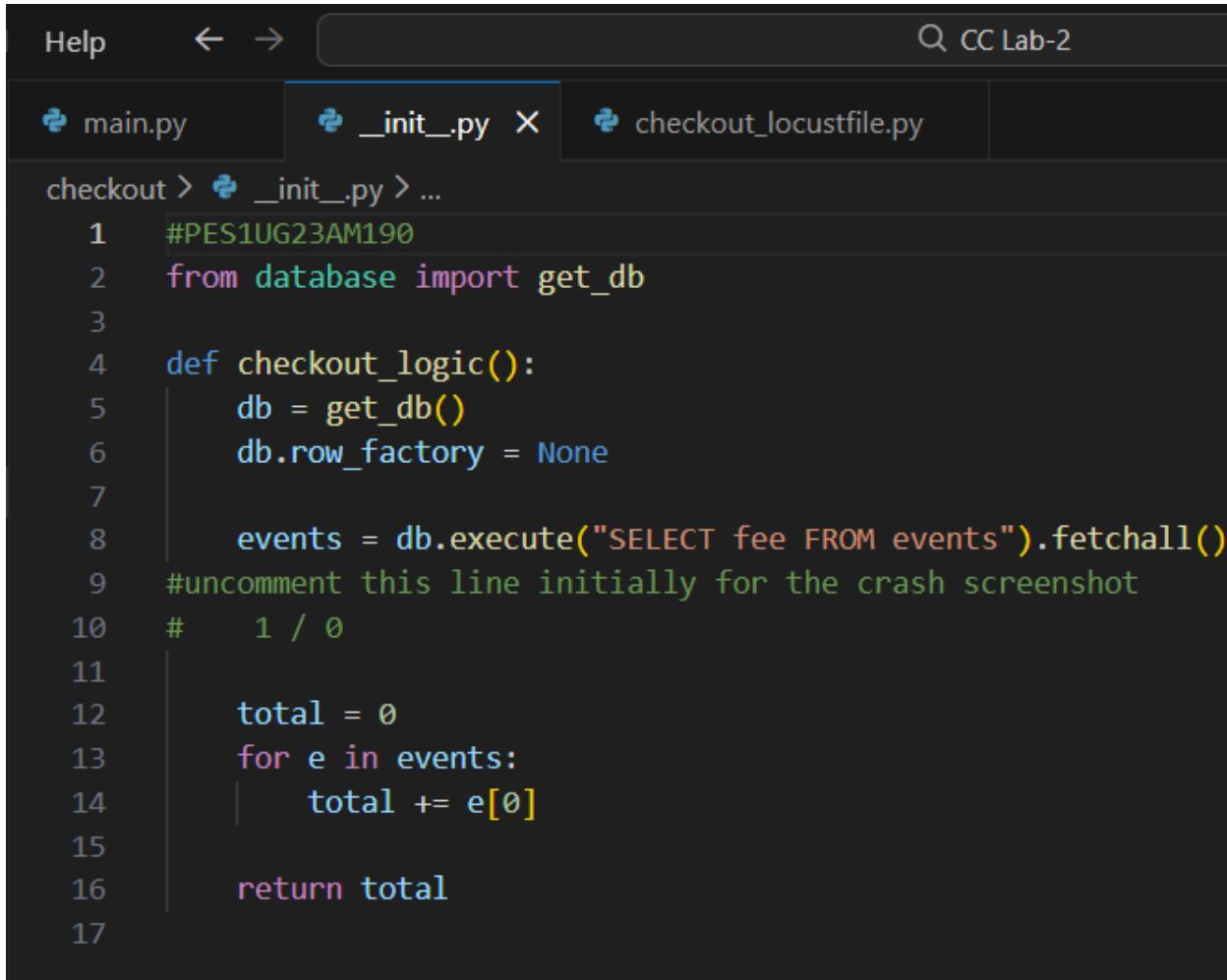
PART 5: Load Testing using Locust

```
C:\COLLEGE\sem6\cc_lab\actual_lab2\PES1UG23AM190\CC Lab-2>.venv\Scripts\activate
(.venv) C:\COLLEGE\sem6\cc_lab\actual_lab2\PES1UG23AM190\CC Lab-2>locust -f locust/checkout_locustfile.py
[2026-01-21 11:52:08,594] NIREN/INFO/locust.main: Starting Locust 2.43.1
[2026-01-21 11:52:08,595] NIREN/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
```

Average response time: 129.2ms



After Code optimisation



The screenshot shows a code editor window with three tabs: 'main.py', '_init_.py' (which is currently selected), and 'checkout_locustfile.py'. The code in '_init_.py' is as follows:

```
1 #PES1UG23AM190
2 from database import get_db
3
4 def checkout_logic():
5     db = get_db()
6     db.row_factory = None
7
8     events = db.execute("SELECT fee FROM events").fetchall()
9 #uncomment this line initially for the crash screenshot
10 #    1 / 0
11
12     total = 0
13     for e in events:
14         total += e[0]
15
16     return total
17
```

SS5: Reduced average response time: 122.57ms

The screenshot shows a dual-monitor setup. The left monitor displays the Locust web interface at localhost:8089. The interface has a dark theme with a navigation bar and tabs for STATISTICS, CHARTS, FAILURES, EXCEPTIONS, CURRENT RATIO, and DOWN. The STATISTICS tab is active, showing a table of performance metrics for a 'checkout' endpoint. The right monitor shows two Command Prompt windows. The top window lists performance statistics for the 'checkout' endpoint, including request counts, failure rates, and response time percentiles. The bottom window shows the command used to run the Locust test.

Locust Statistics (localhost:8089)

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)
GET	//checkout	18	0	7	2100	2100	122.57	6	2100
	Aggregated	18	0	7	2100	2100	122.57	6	2100

Performance Metrics (Approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%	99.9%	99.99%
GET	//checkout	7	7	7	8	10	2100	2100	2100	2100
	Aggregated	7	7	7	8	10	2100	2100	2100	2100

Command Prompt - uvicorn n

```
uvicorn n:app --host 0.0.0.0 --port 8089
```

Command Prompt

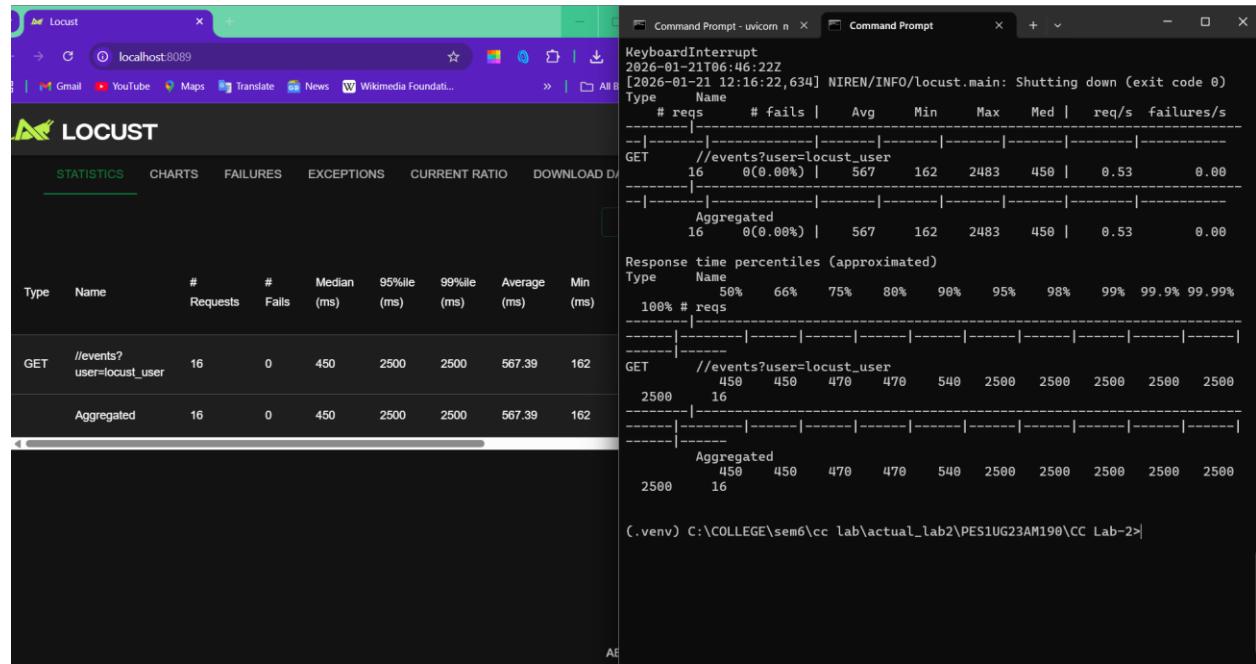
```
locust -f locustfile.py -u 10 -r 10 -t 60s
```

(.venv) C:\COLLEGE\sem6\cc_lab\actual_lab2\PES1UG23AM190\CC_Lab-2>

PART 7: Optimise events and my_events(DIY)

Route 1: /events

SS6: Screenshot BEFORE optimization(Avg response time:567.39ms)



Code before optimization:

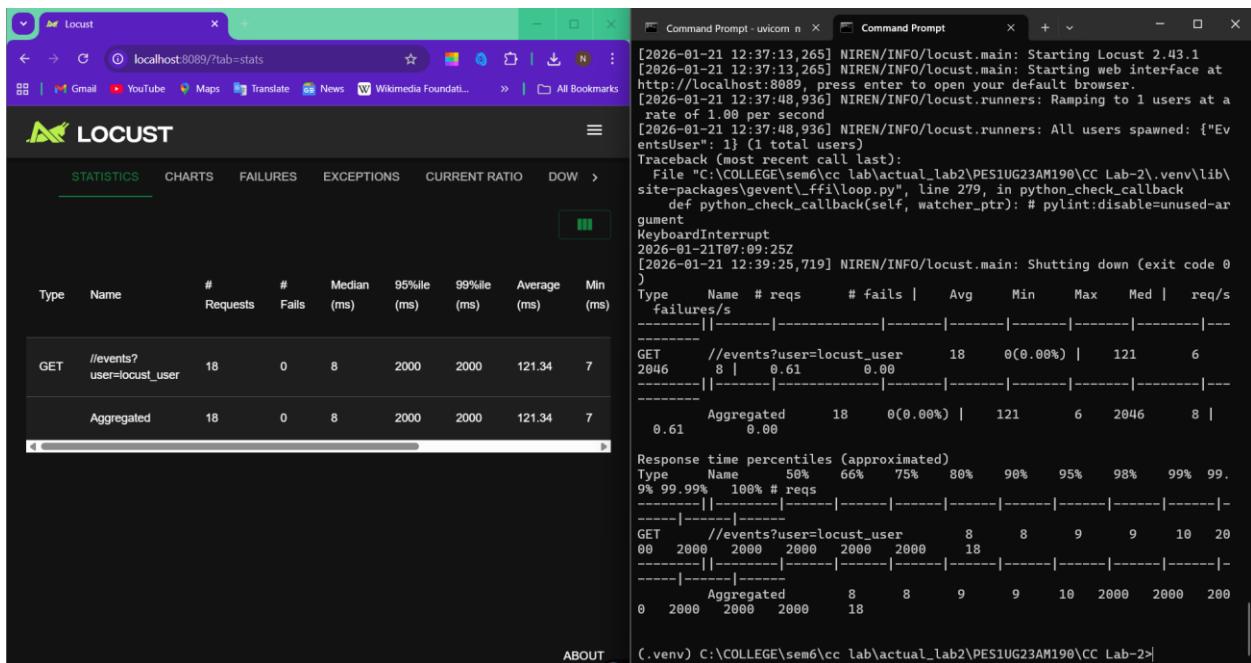
The screenshot shows a code editor window titled "CC Lab-2". The main pane displays a Python script named "main.py". The code implements a login endpoint and an "events" endpoint. The "events" endpoint retrieves data from a database and includes a performance optimization loop that adds a large number to a variable "waste". The code uses Jinja2 templating to render an "events.html" template.

```
Help      ← →      CC Lab-2
main.py  X  __init__.py  myevents_locustfile.py  insert_events.py  checkout_locustfile.py
main.py > events
43     def login(request: Request, username: str = Form(...), password: str = Form(...)):
53         {"request": request, "error": "X Invalid username or password", "user": ""}
54     )
55
56     return RedirectResponse(f"/events?user={username}", status_code=302)
57
58
59 #PES1UG23AM190
60 @app.get("/events", response_class=HTMLResponse)
61 def events(request: Request, user: str):
62     db = get_db()
63     rows = db.execute("SELECT * FROM events").fetchall()
64
65     waste = 0
66     for i in range(3000000):
67         waste += i % 3
68
69     return templates.TemplateResponse(
70         "events.html",
71         {"request": request, "events": rows, "user": user}
72     )
73
```

After optimizing code:

```
Help      ← →      Q CC Lab-2
main.py  X  _init_.py  myevents_locustfile.py  insert_events.py  checkout
main.py > ...
43     def login(request: Request, username: str) -> RedirectResponse:
52         "login.html",
53         {"request": request, "error": None}
54     )
55
56     return RedirectResponse(f"/events?user={username}", status_code=302)
57
58
59 #PES1UG23AM190
60 @app.get("/events", response_class=HTMLResponse)
61 def events(request: Request, user: str):
62     db = get_db()
63     rows = db.execute("SELECT * FROM events").fetchall()
64
65     #waste = 0
66     #for i in range(3000000):
67     #    waste += i % 3
68
69     return templates.TemplateResponse(
70         "events.html",
71         {"request": request, "events": rows, "user": user}
72     )
73
74
```

SS7: Screenshot AFTER optimization(Avg response time: 121.34 ms)



Question and answers for Route 1:

1) What was the bottleneck?

Ans: The /events route contained an unnecessary computation loop that performed millions of iterations, causing artificial CPU delay and increasing response time under load.

2) What change did you make?

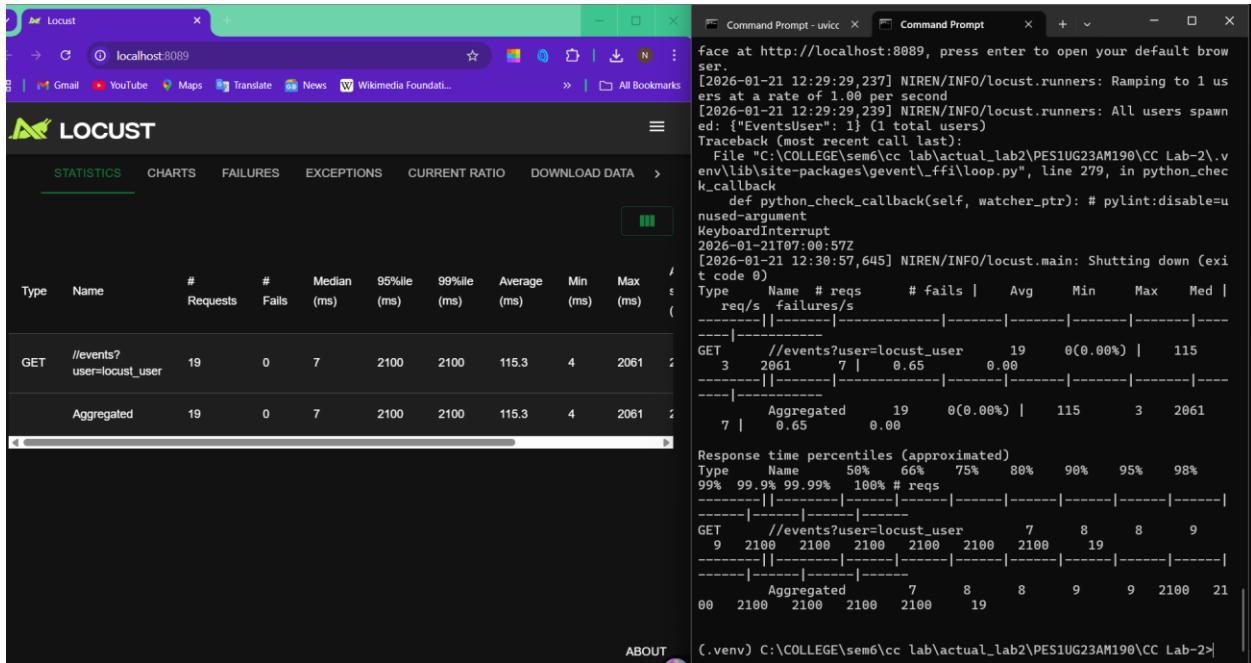
Ans: The redundant loop was removed so the route only performs the required database query.

3) Why did the performance improve?

Ans: Removing the unnecessary computation reduced CPU usage and request processing time, allowing the server to handle more concurrent requests efficiently, thus reducing Average response time.

Route 2: /my-events

SS8: Screenshot BEFORE optimization (Avg response time:115.3 ms)



Code before optimization:

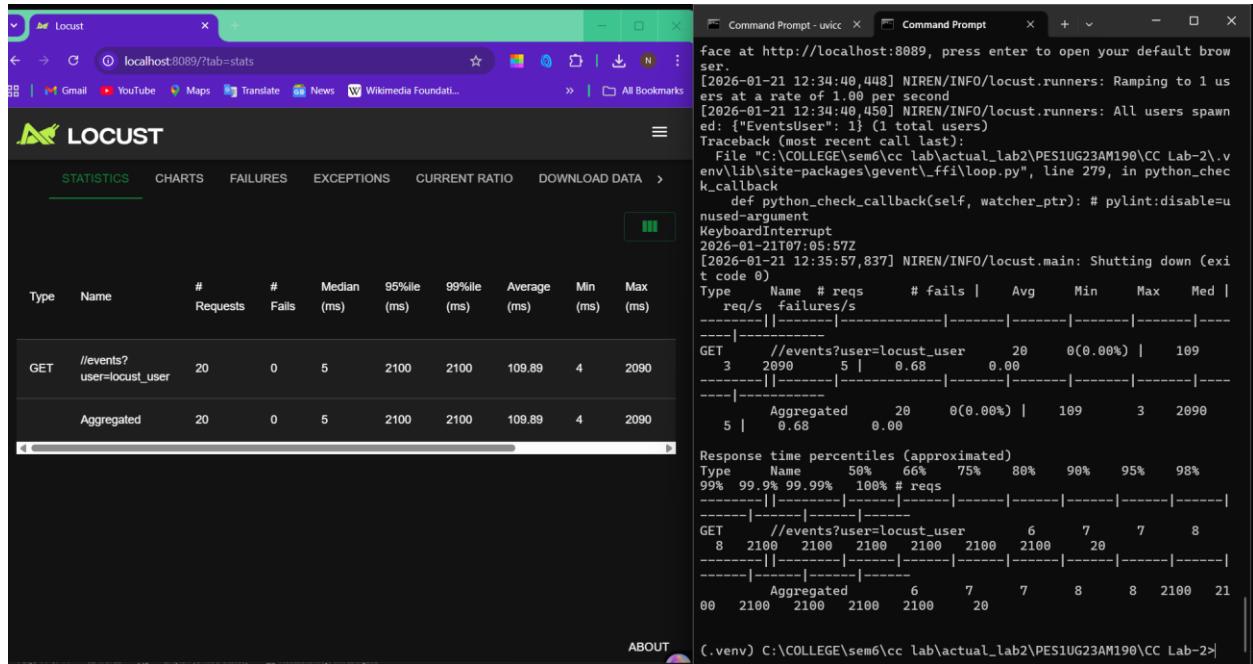
The screenshot shows a code editor interface with a dark theme. The title bar reads "CC Lab-2". The left sidebar lists files: "main.py" (selected), "_init_.py", "myevents_locustfile.py", "insert_events.py", and "checkout_locustfile.py". The main pane displays Python code for a web application:

```
76     def register_event(event_id: int, user: str):
77         1 / 0
78
79         db = get_db()
80         db.execute("INSERT INTO registrations VALUES (?,?)", (user, event_id))
81         db.commit()
82
83
84         return RedirectResponse(f"/my-events?user={user}", status_code=302)
85
86 #PES1UG23AM190
87 @app.get("/my-events", response_class=HTMLResponse)
88 def my_events(request: Request, user: str):
89     db = get_db()
90     rows = db.execute(
91         """
92             SELECT events.name, events.fee
93             FROM events
94             JOIN registrations ON events.id = registrations.event_id
95             WHERE registrations.username=?
96             """,
97             (user,)
98         ).fetchall()
99
100
101    dummy = 0
102    for _ in range(1500000):
103        dummy += 1
104
105    return templates.TemplateResponse(
106        "my_events.html",
107        {"request": request, "events": rows, "user": user}
108    )
109
```

After optimizing code:

```
Help < > CC Lab-2
main.py _init_.py myevents_locustfile.py insert_events.py checkout_loc
main.py > my_events
76     def register_event(event_id: int, user| @app.get("/events") Aa ab .* No results
78         | 1 / 0
79
80         db = get_db()
81         db.execute("INSERT INTO registrations VALUES (?,?)", (user, event_id))
82         db.commit()
83
84         return RedirectResponse(f"/my-events?user={user}", status_code=302)
85
86 #PES1UG23AM190
87 @app.get("/my-events", response_class=HTMLResponse)
88 def my_events(request: Request, user: str):
89     db = get_db()
90     rows = db.execute(
91         """
92             SELECT events.name, events.fee
93             FROM events
94             JOIN registrations ON events.id = registrations.event_id
95             WHERE registrations.username=?
96             """,
97             (user,)
98         ).fetchall()
99
100
101    #dummy = 0
102    #for _ in range(1500000):
103    |  #dummy += 1
104
105    return templates.TemplateResponse(
106        "my_events.html",
107        {"request": request, "events": rows, "user": user}
108    )
109
```

SS9: Screenshot AFTER optimization(Avg response time:109.89ms)



Question and answers for Route 2:

1) What was the bottleneck?

Ans: The /my-events route had a dummy loop that introduced an intentional delay after fetching data from the database, slowing down every request.

2) What change did you make?

Ans: The dummy delay loop was removed, leaving only the essential database join query and response rendering.

3) Why did the performance improve?

Ans: Eliminating the artificial delay reduced response latency and improved throughput, resulting in faster responses and better performance under load, thus reducing Average response time.