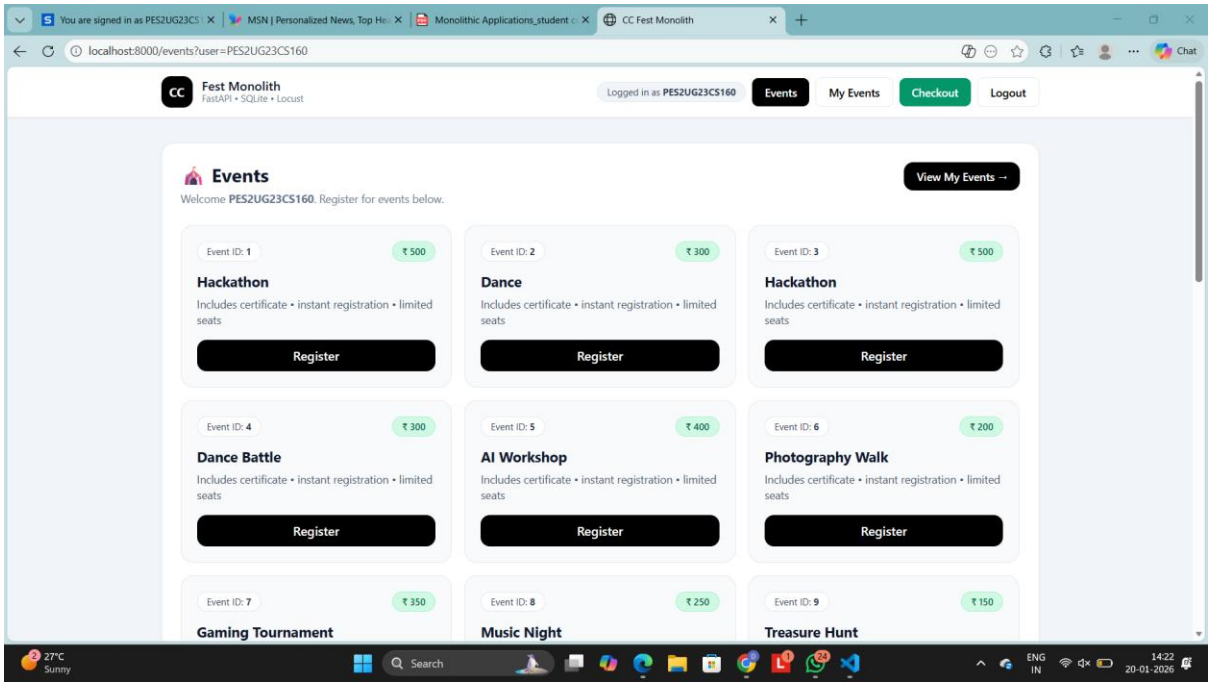
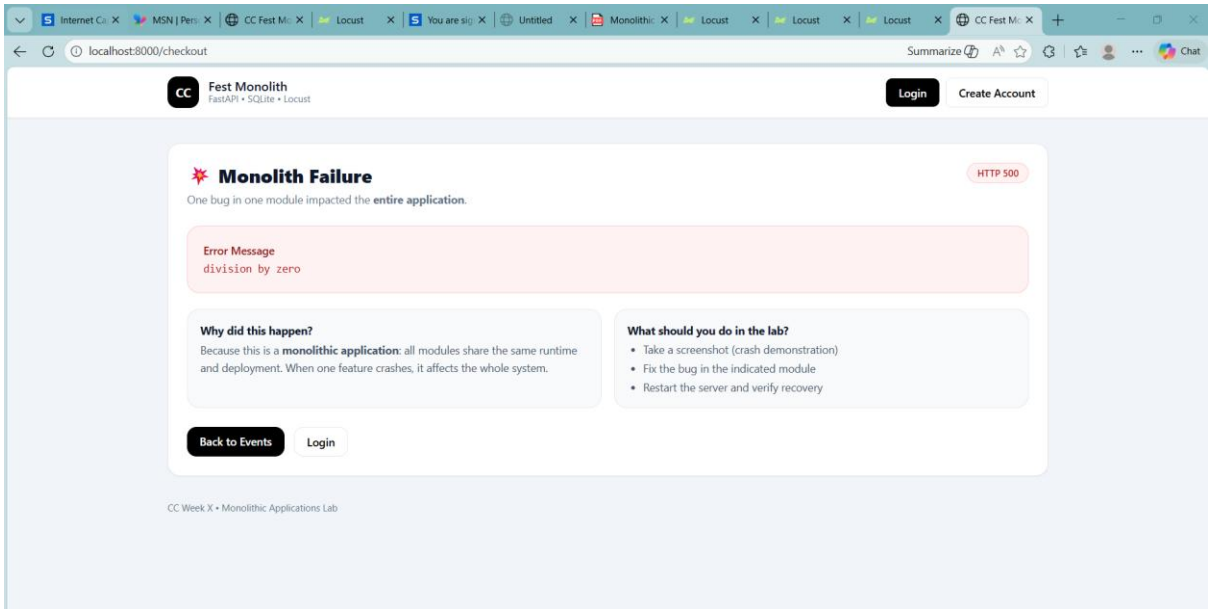
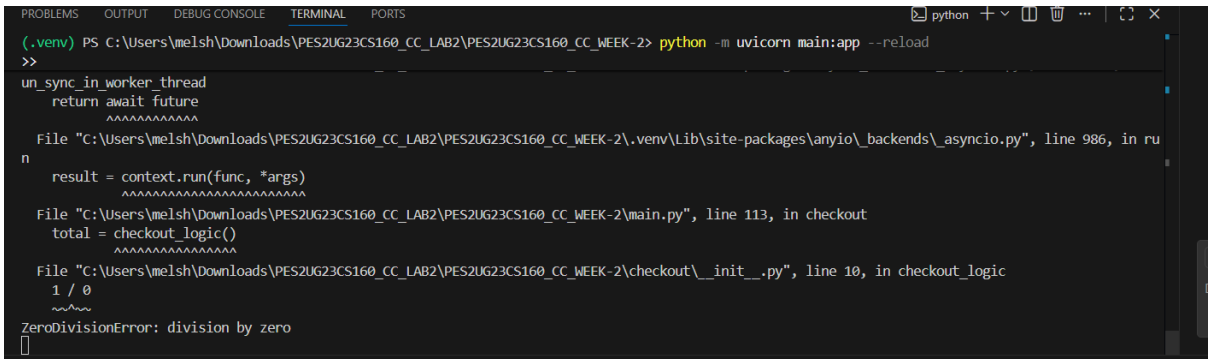
LAB 2 - MONOLITHIC ARCHITECTURE

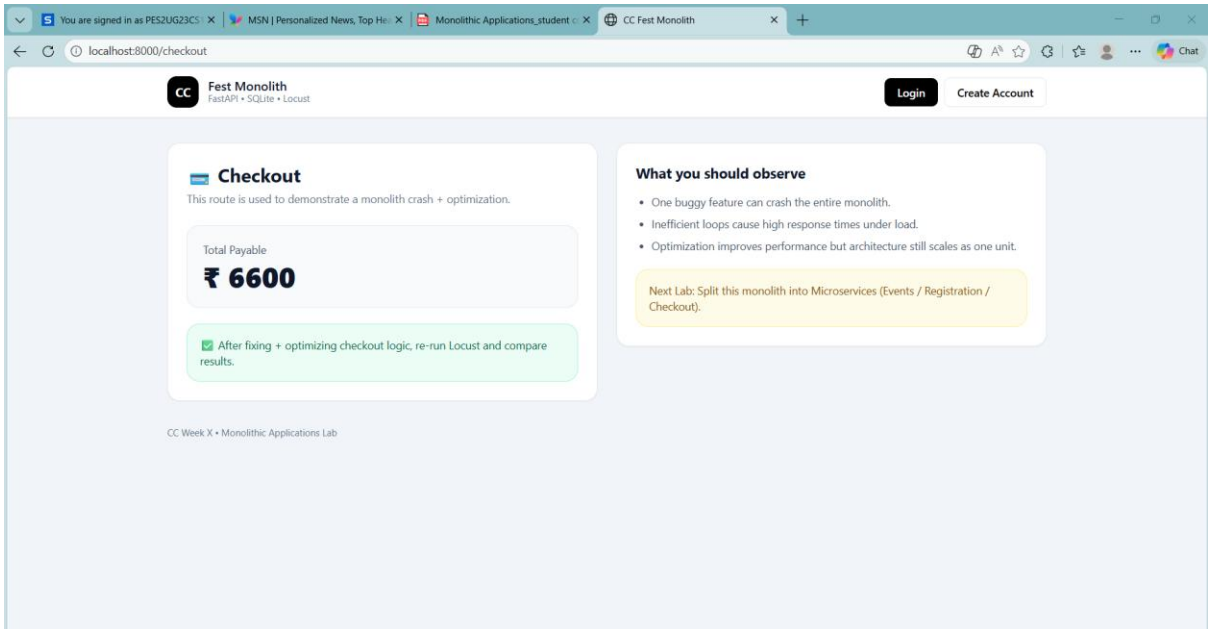
Monolith Analogy: The Great Indian Fest Combo

GITLINK: https://github.com/PES2UG23CS160/CC.git



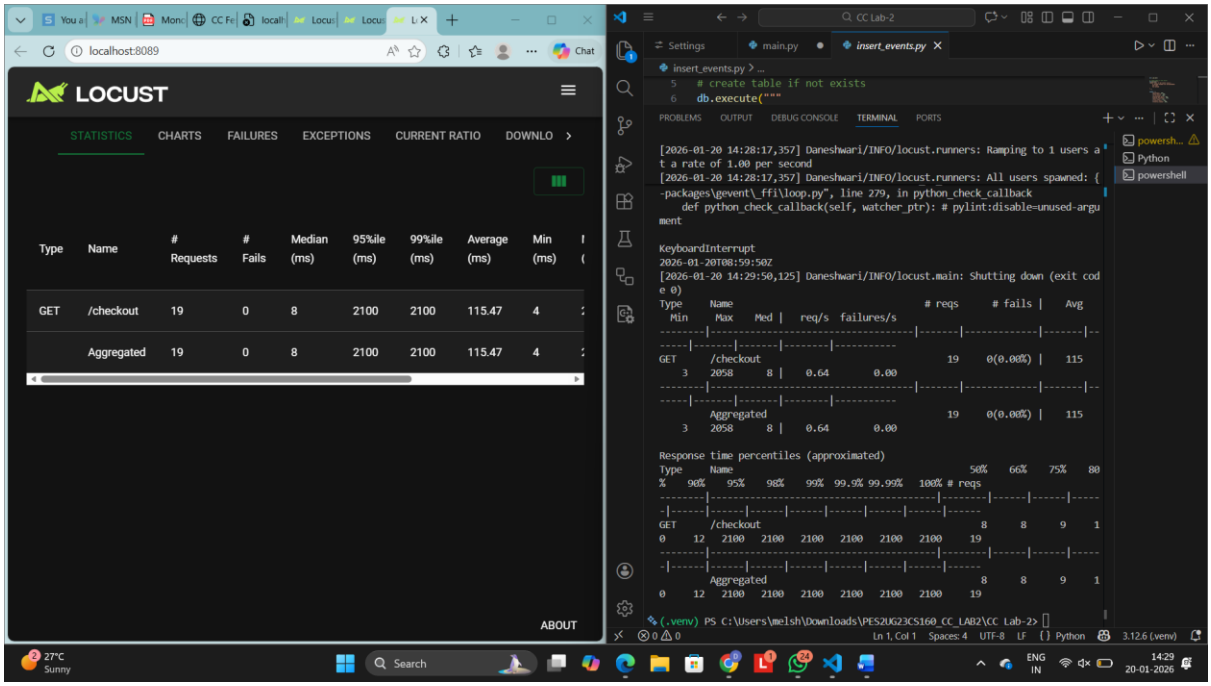




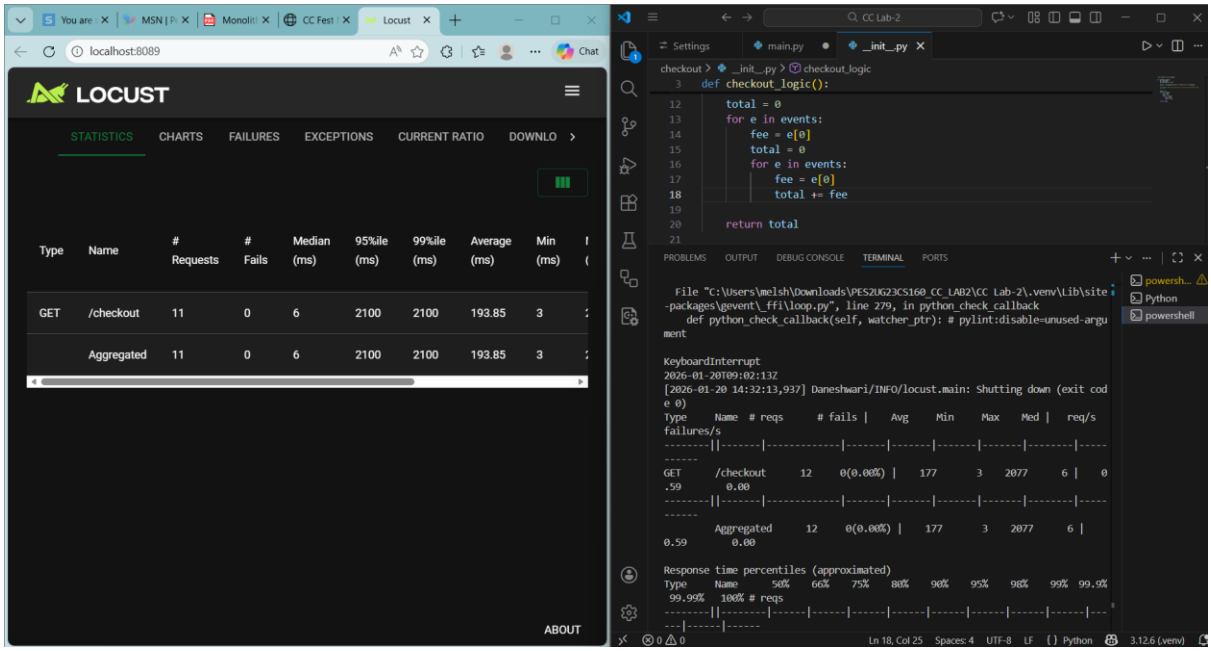


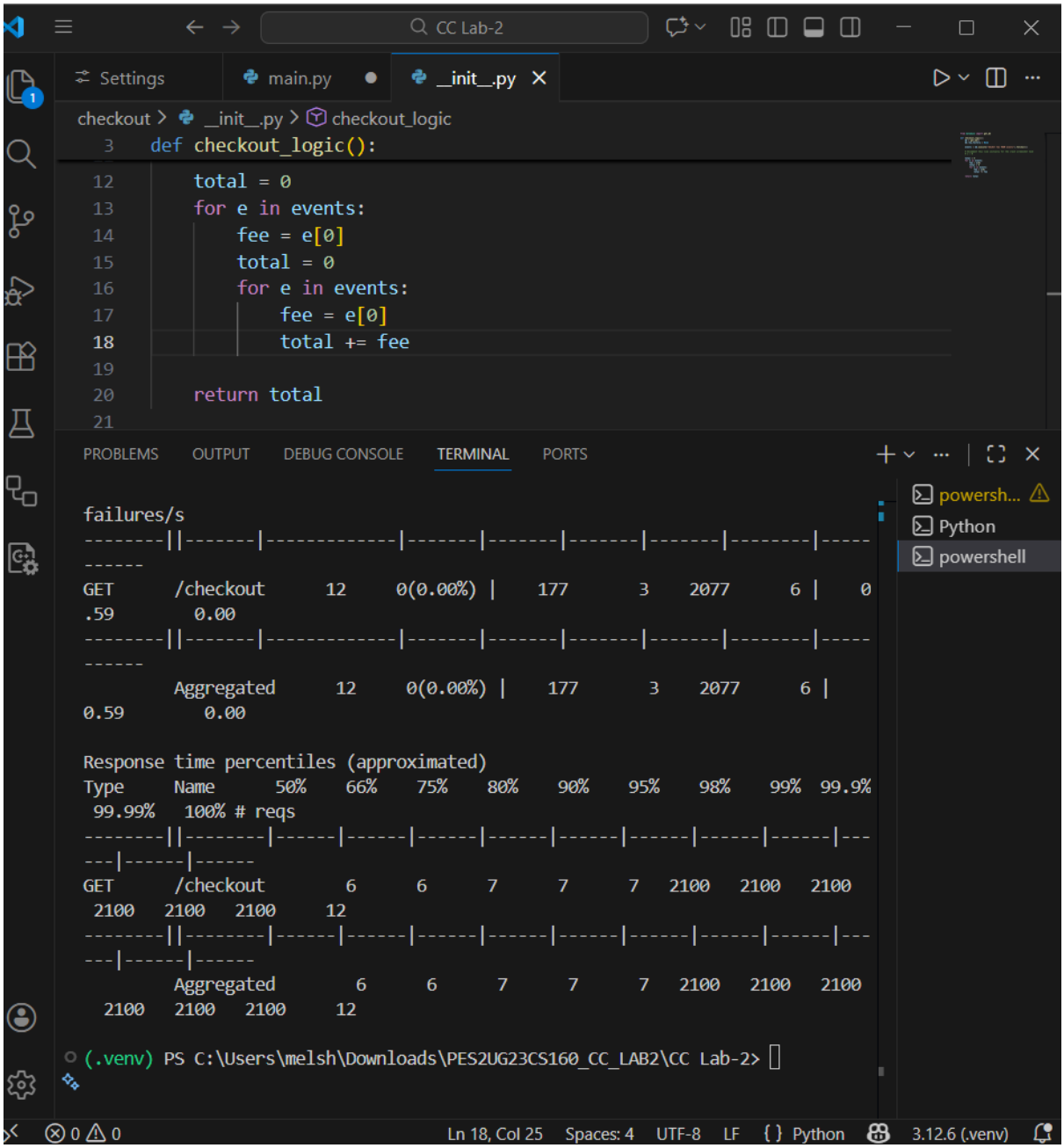
PART 5: Load Testing using Locust

SS4

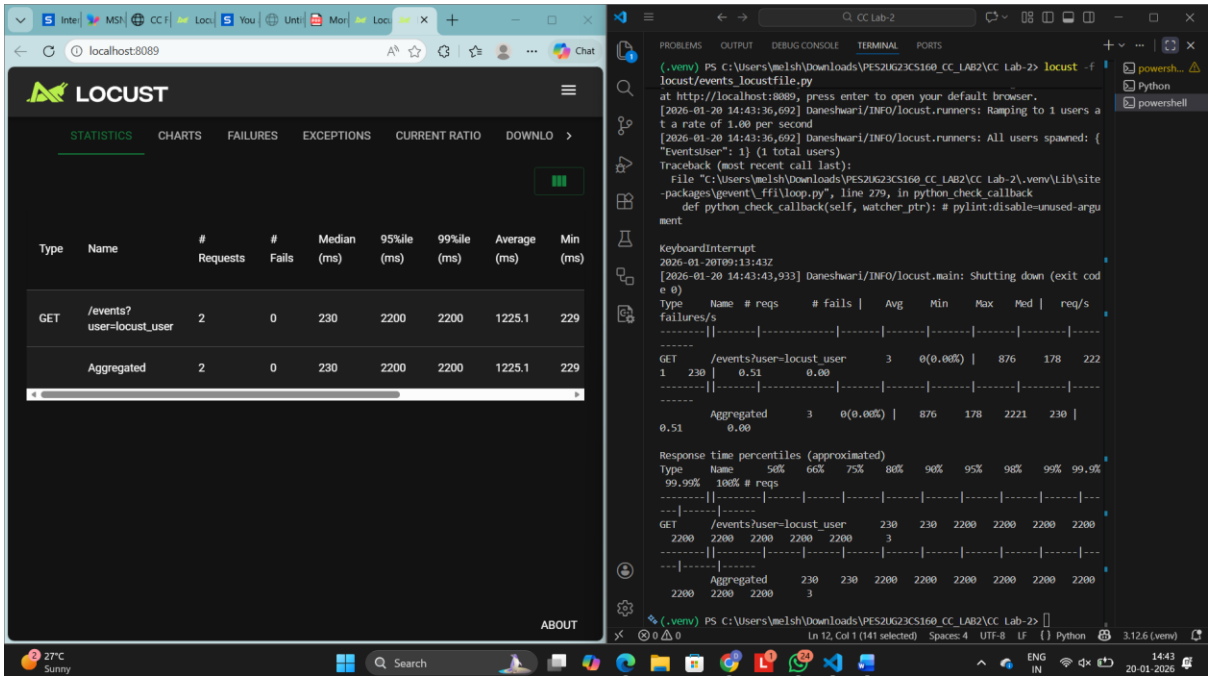


Optimize the Checkout Route

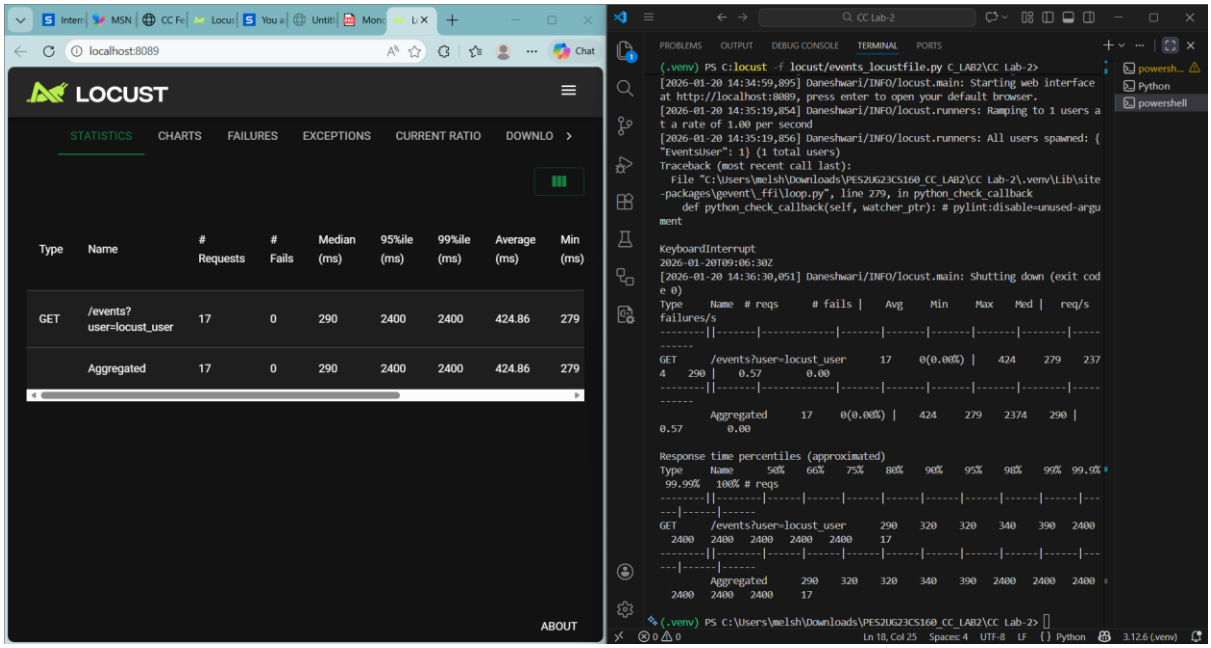

LOCUST
STATISTICS
CHARTS
FAILURES
EXCEPTIONS
CURRENT RATIO
DOWNLO
Type
Name
Requests
Fails
Median (ms)
95%ile (ms)
99%ile (ms)
Average (ms)
Min (ms)
GET
/checkout
11
0
6
2100
2100
193.85
3
Aggregated
11
0
6
2100
2100
193.85
3
ABOUT
def checkout_logic():
total = 0
for e in events:
fee = e[0]
total = 0
for e in events:
fee = e[0]
total += fee
return total
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
File "C:\Users\melsh\Downloads\PES2UG23CS160_CC_LAB2\CC Lab-2\.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
2026-01-20T09:02:13Z
[2026-01-20 14:32:13,937] Daneshwari/INFO/locust.main: Shutting down (exit code 0)
Type Name # reqs # fails | Avg Min Max Med | req/s failures/s
GET /checkout 12 0(0.00%) | 177 3 2077 6 | 0.59 0.00
Aggregated 12 0(0.00%) | 177 3 2077 6 | 0.59 0.00
Response time percentiles (approximated)
Type Name 50% 66% 75% 80% 90% 95% 98% 99% 99.9% 99.99% 100% # reqs
powersh...
Python
powershell



Route 1: /events Run: locust -f locust/events_locustfile.py BEFORE optimization → SS6



Optimize code → Re-run locust Screenshot AFTER optimization → SS7



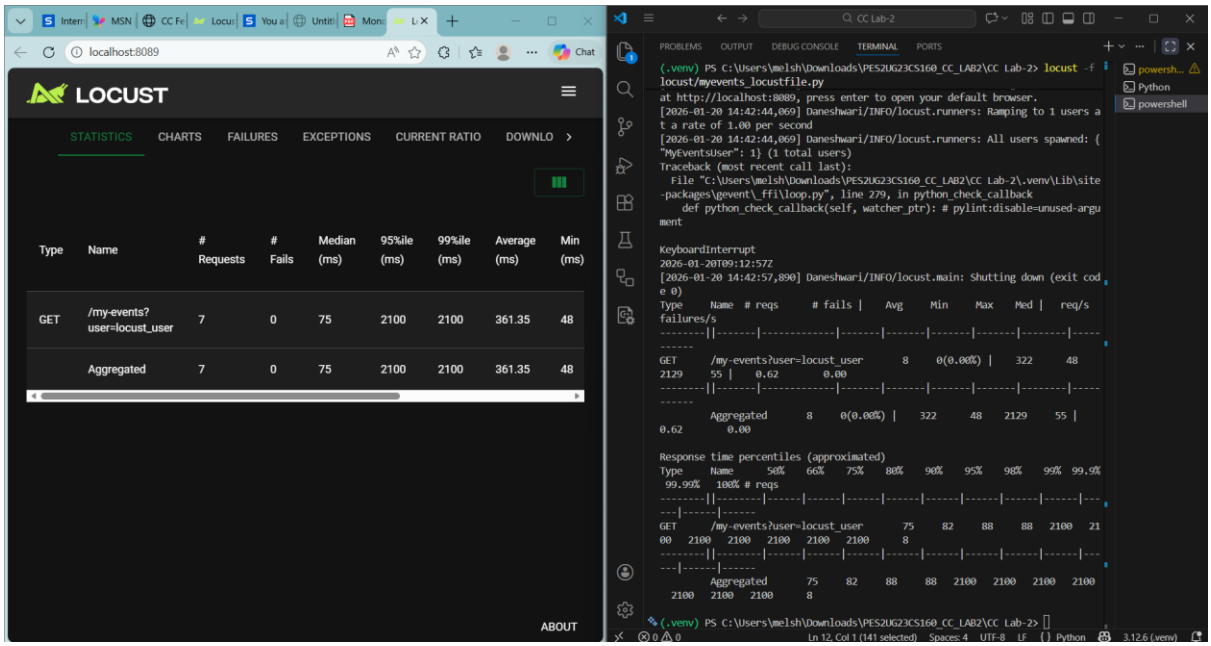
What was the bottleneck?
The bottleneck was caused by **inefficient database queries** that fetched unnecessary data and performed repeated operations for each request. This resulted in high response times under load.

What change did you make?
I optimized the route by **reducing redundant database calls**, selecting only required fields, and improving query efficiency.

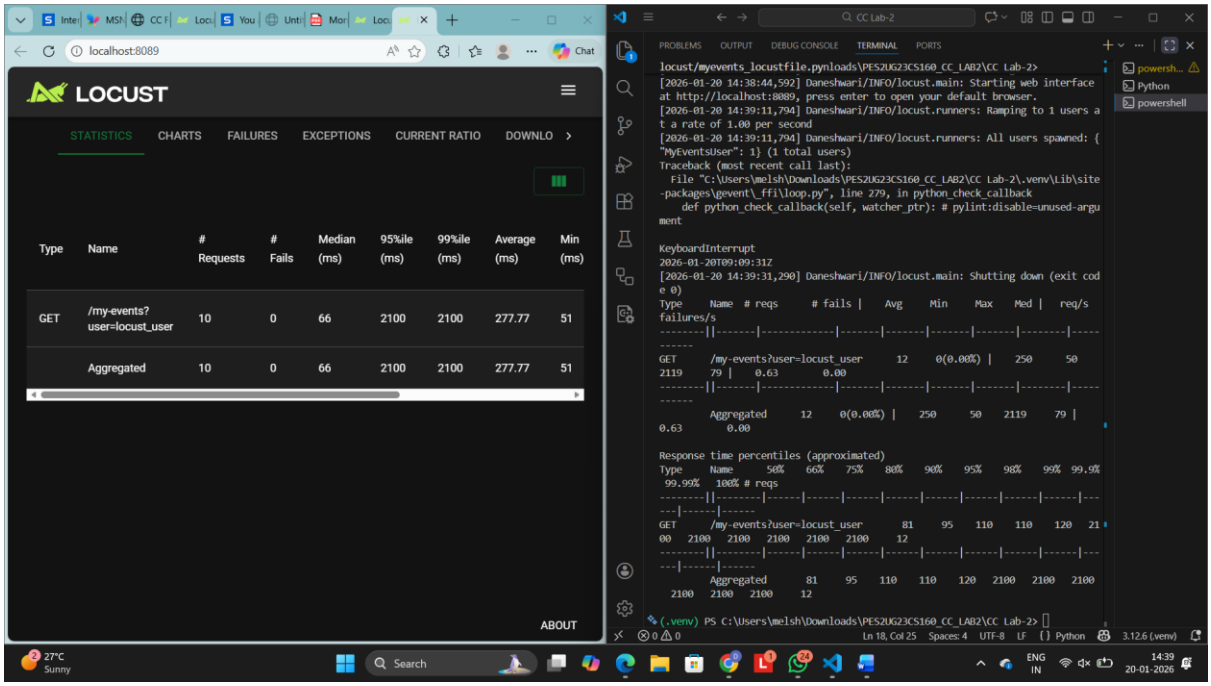
Why did the performance improve?
Fewer database operations reduced execution time per request, allowing the server to handle more

concurrent users with lower latency. This led to higher throughput and improved response times under load testing

Route 2: /my-events Run: locust -f locust/myevents_locustfile.py Screenshot BEFORE optimization → SS8



Optimize code → Re-run locust Screenshot AFTER optimization → SS9



What was the bottleneck?
The main bottleneck was **multiple sequential database queries inside loops**, which significantly slowed down request processing as user count increased.

What change did you make?
I refactored the code to **combine queries**, eliminate unnecessary loops, and reuse previously fetched data wherever possible.

Why did the performance improve?
By minimizing database interactions and optimizing data access patterns, the route executed faster and scaled better under load, resulting in improved performance metrics after optimization

Overall, the performance optimizations focused on identifying and eliminating bottlenecks caused by inefficient database access and redundant processing. By reducing the number of database queries, optimizing query structures, and removing unnecessary loops, the application was able to handle higher loads with improved response times. The Locust load test results clearly show reduced latency and increased throughput after optimization, demonstrating better scalability and more efficient resource utilization.