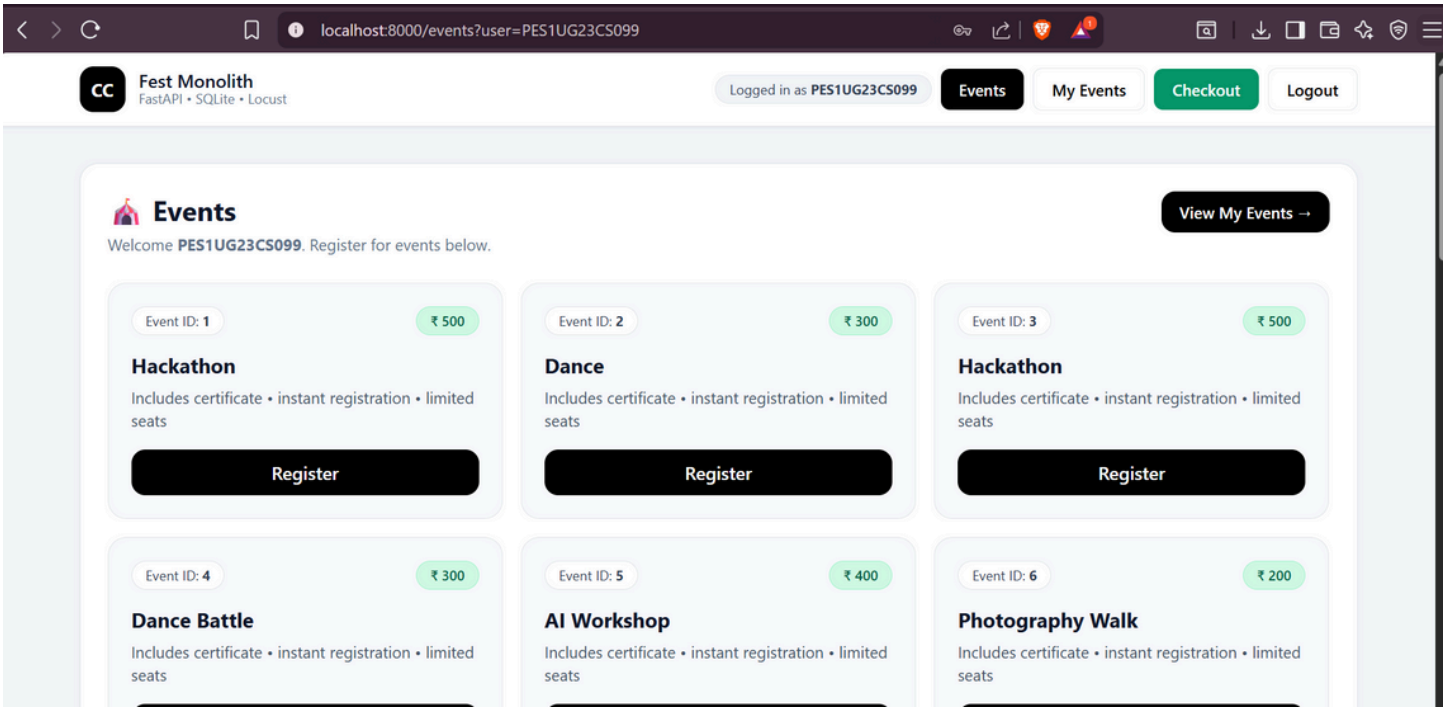
CC LAB-2

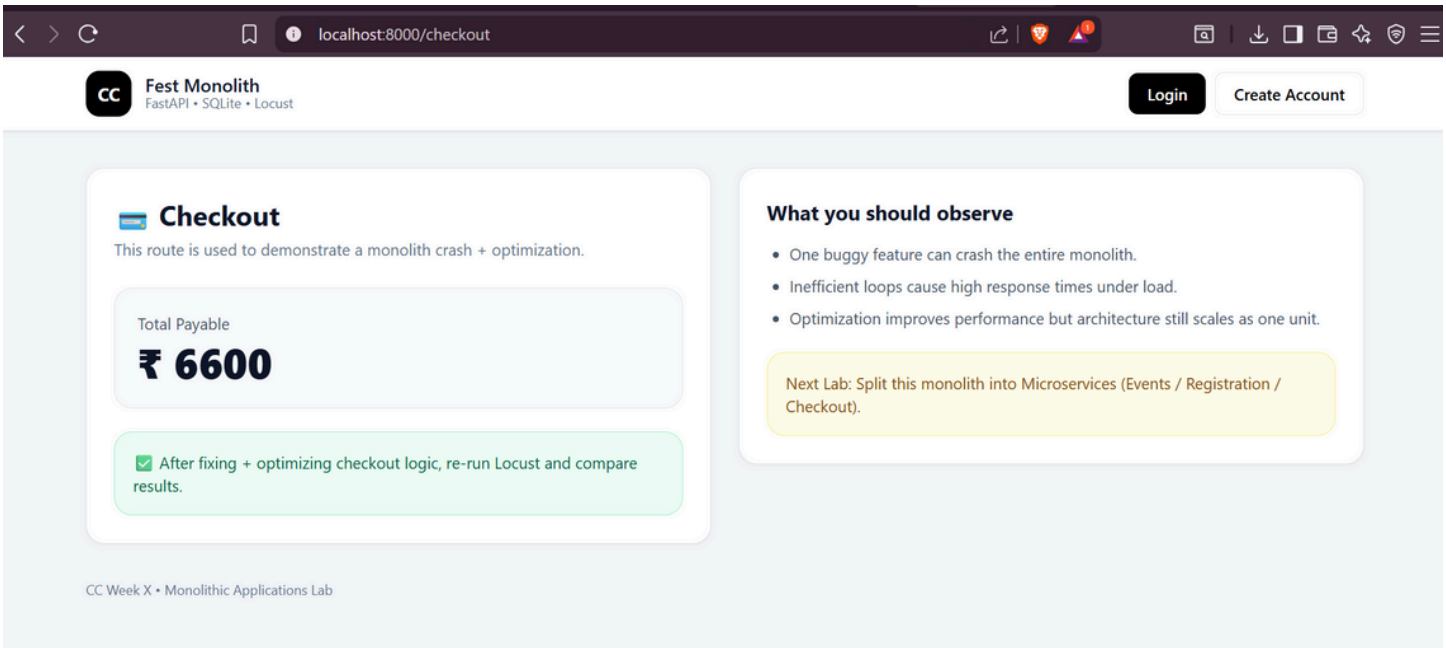
ARCHI PANKAJ

PES1UG23CS099

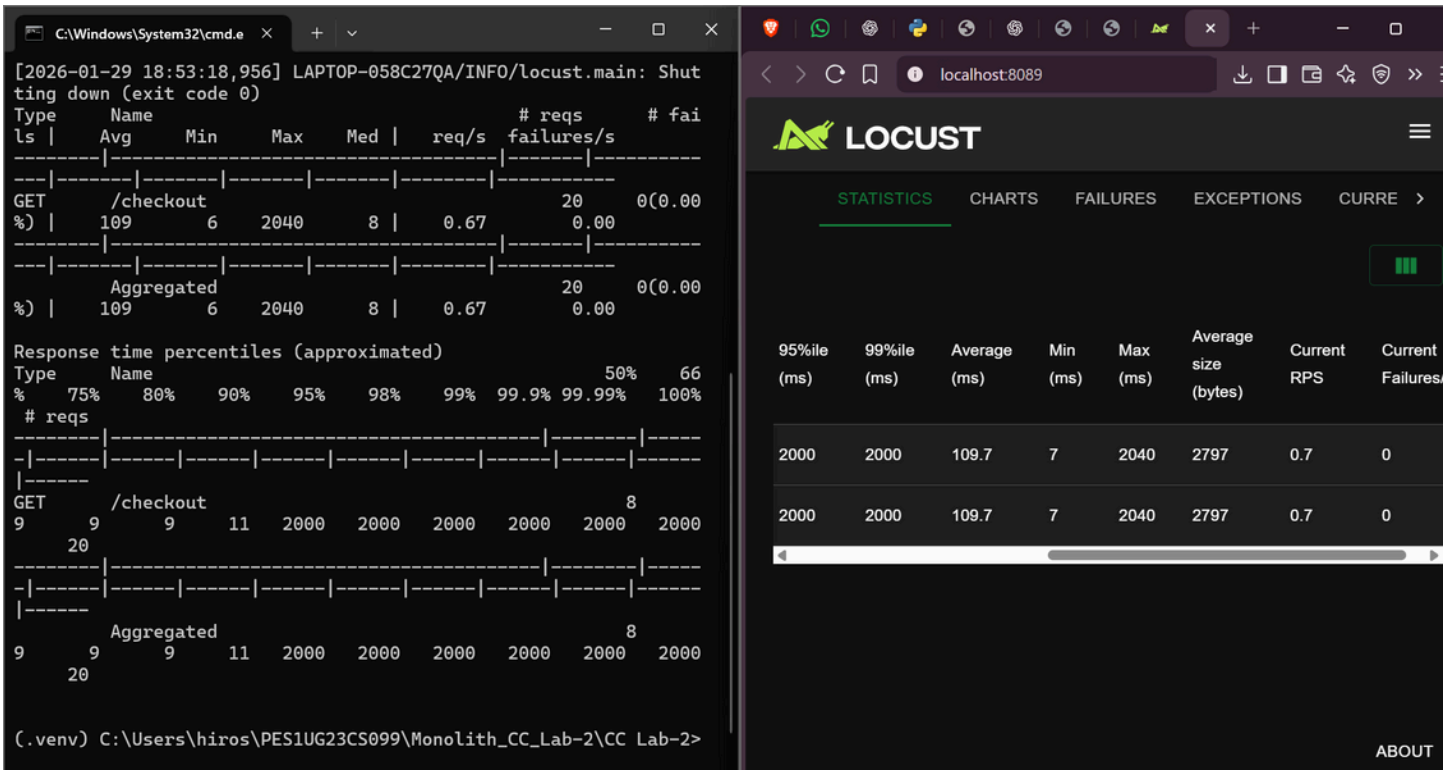
SS1:

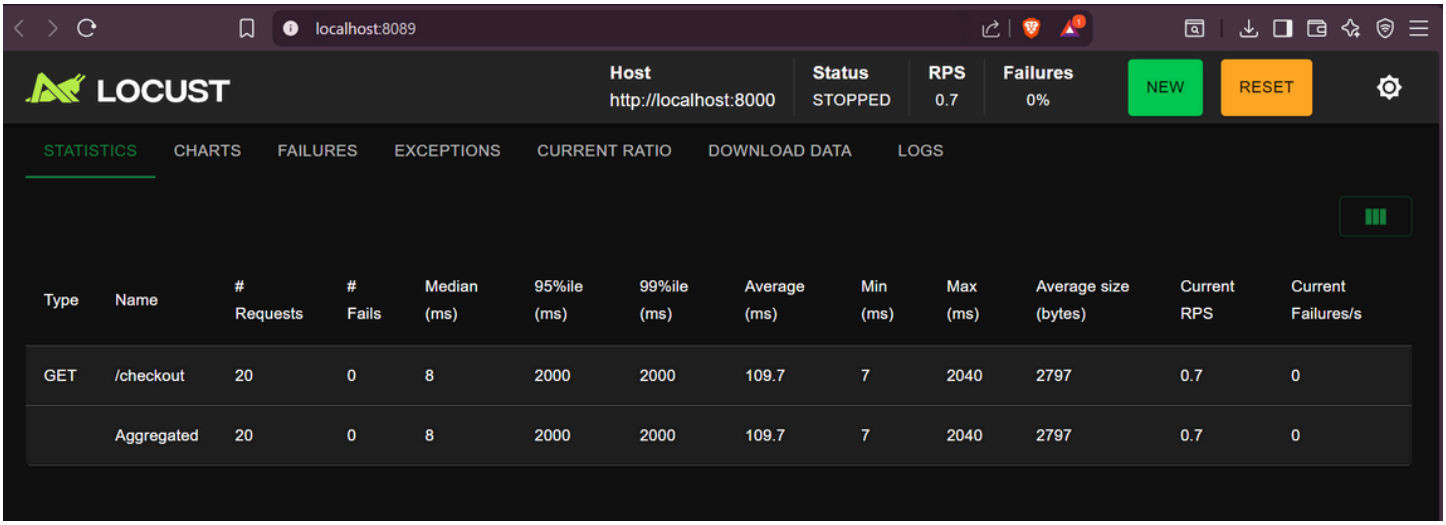


SS2:

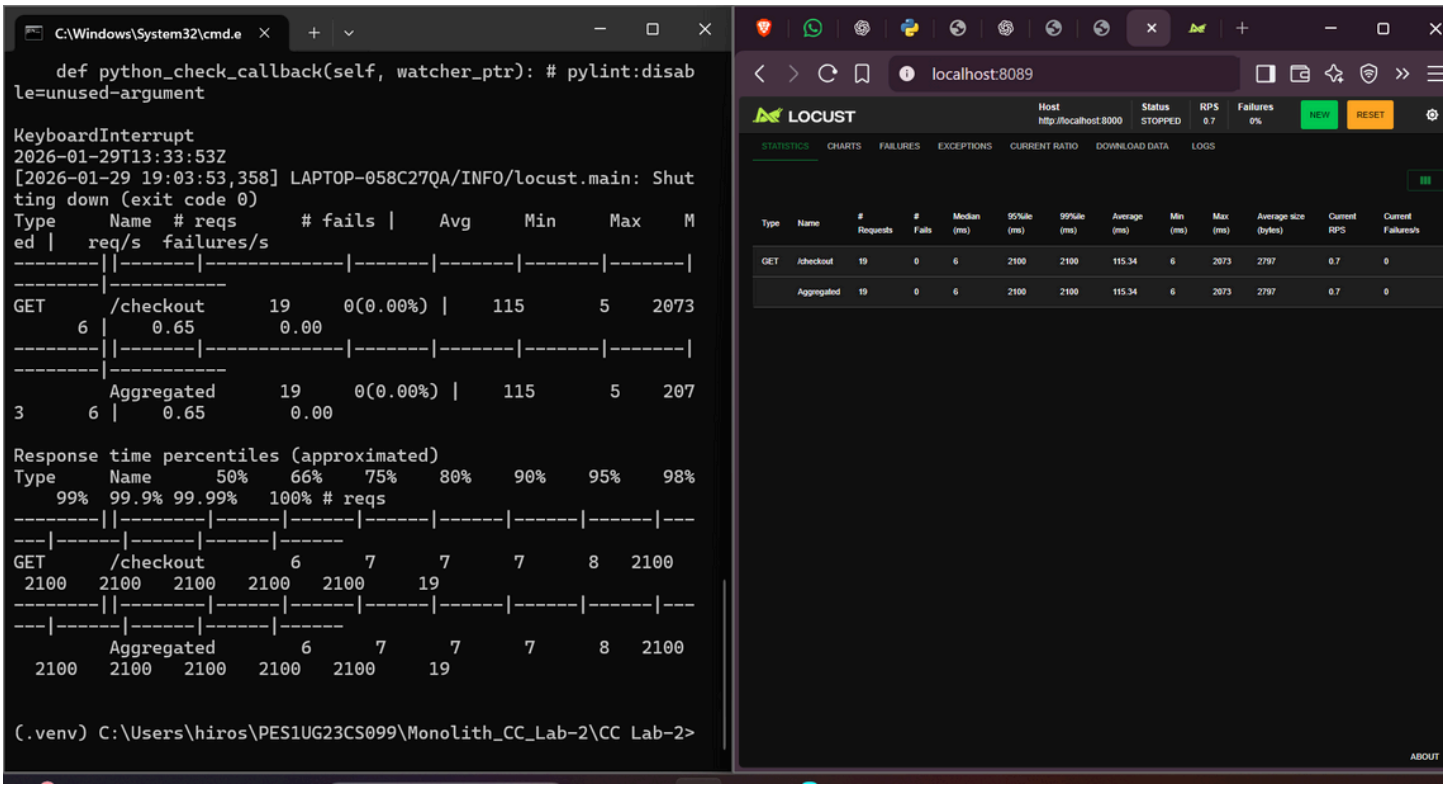


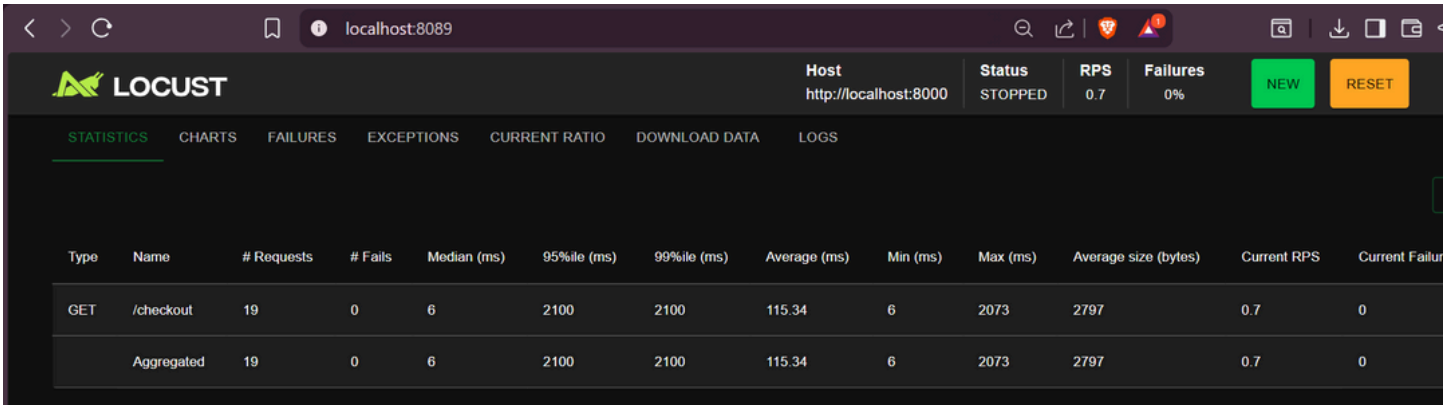
SS4:before


C:\Windows\System32\cmd.e
[2026-01-29 18:53:18,956] LAPTOP-058C27QA/INFO/locust.main: Shut
ting down (exit code 0)
Type Name # reqs # fai
ls | Avg Min Max Med | req/s failures/s
GET /checkout 20 0(0.00
%) | 109 6 2040 8 | 0.67 0.00
Aggregated 20 0(0.00
%) | 109 6 2040 8 | 0.67 0.00
Response time percentiles (approximated)
Type Name 50% 66
% 75% 80% 90% 95% 98% 99% 99.9% 99.99% 100%
# reqs
GET /checkout 8
9 9 9 11 2000 2000 2000 2000 2000 2000
20
Aggregated 8
9 9 9 11 2000 2000 2000 2000 2000 2000
20
(.venv) C:\Users\hiros\PES1UG23CS099\Monolith_CC_Lab-2\CC Lab-2>
localhost:8089
LOCUST
STATISTICS
CHARTS
FAILURES
EXCEPTIONS
CURRE
95%ile (ms)
99%ile (ms)
Average (ms)
Min (ms)
Max (ms)
Average size (bytes)
Current RPS
Current Failures
2000 2000 109.7 7 2040 2797 0.7 0
2000 2000 109.7 7 2040 2797 0.7 0
ABOUT

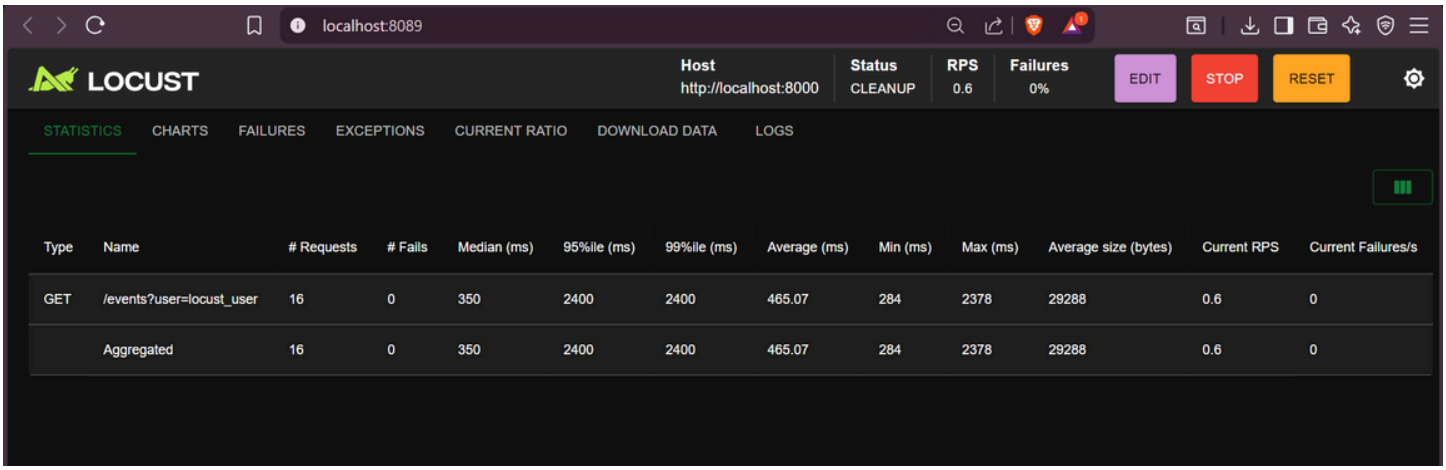

localhost:8089
LOCUST
Host
http://localhost:8000
Status
STOPPED
RPS
0.7
Failures
0%
NEW
RESET
STATISTICS
CHARTS
FAILURES
EXCEPTIONS
CURRENT RATIO
DOWNLOAD DATA
LOGS
Type
Name
Requests
Fails
Median (ms)
95%ile (ms)
99%ile (ms)
Average (ms)
Min (ms)
Max (ms)
Average size (bytes)
Current RPS
Current Failures/s
GET /checkout 20 0 8 2000 2000 109.7 7 2040 2797 0.7 0
Aggregated 20 0 8 2000 2000 109.7 7 2040 2797 0.7 0

SS5:after

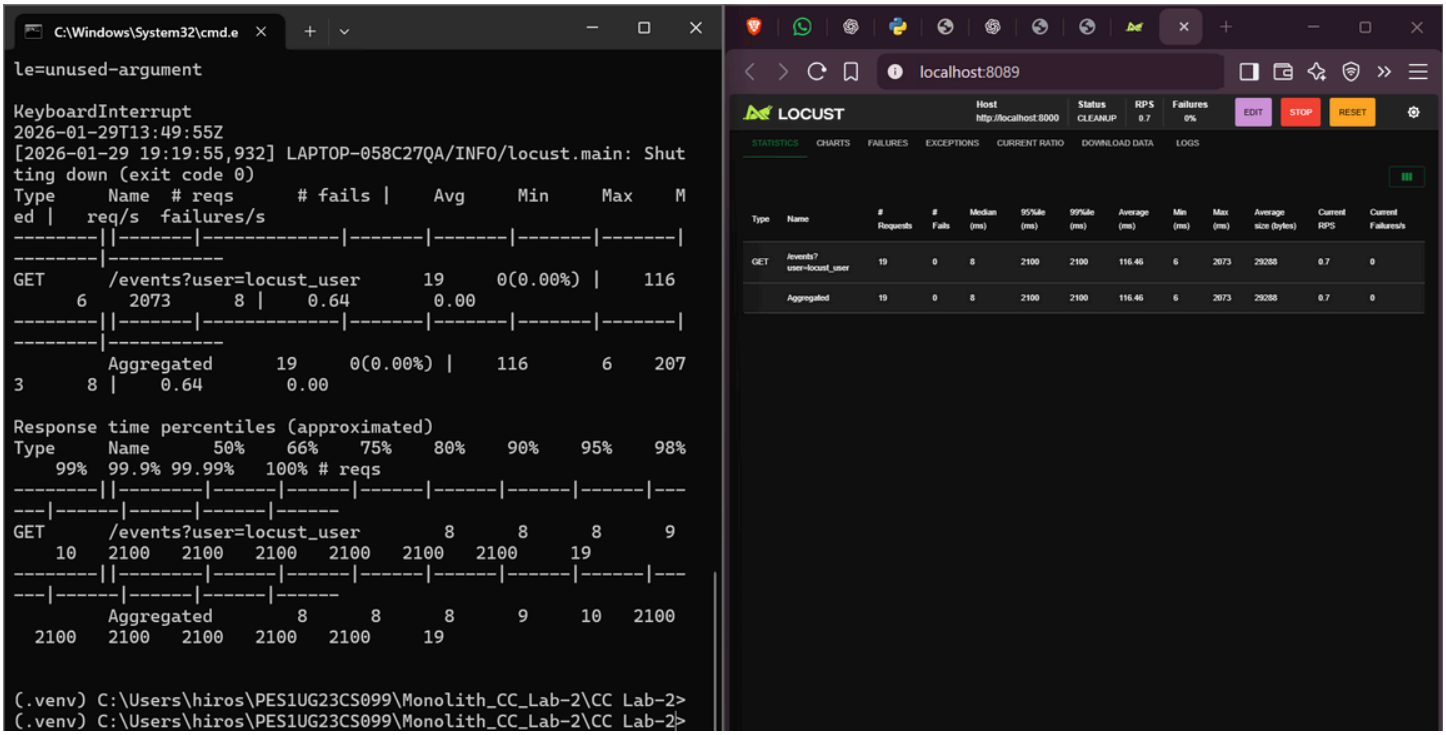


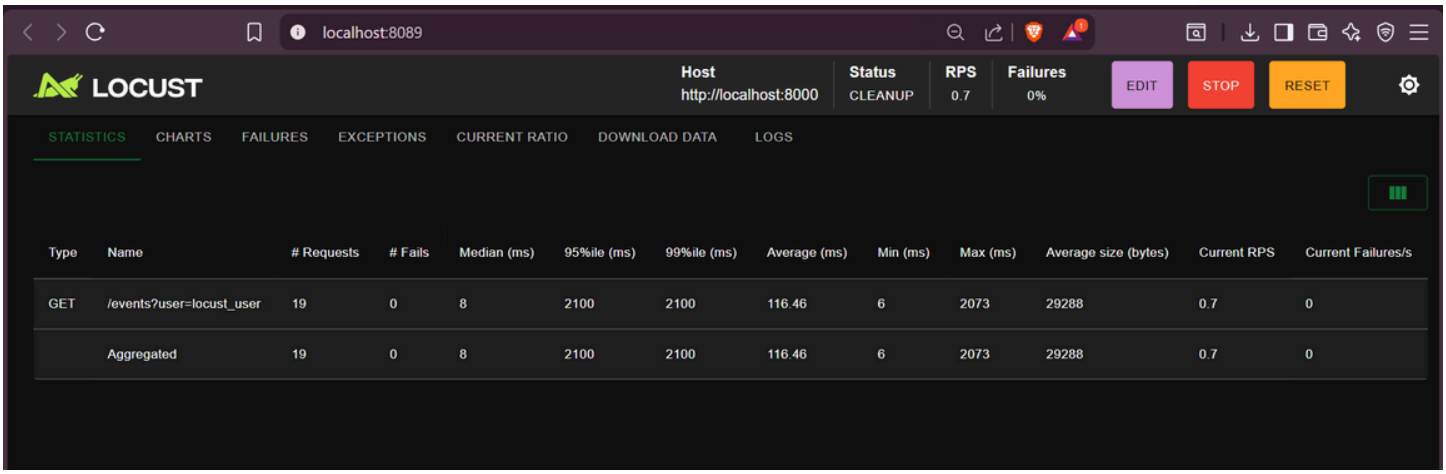


SS6:before events

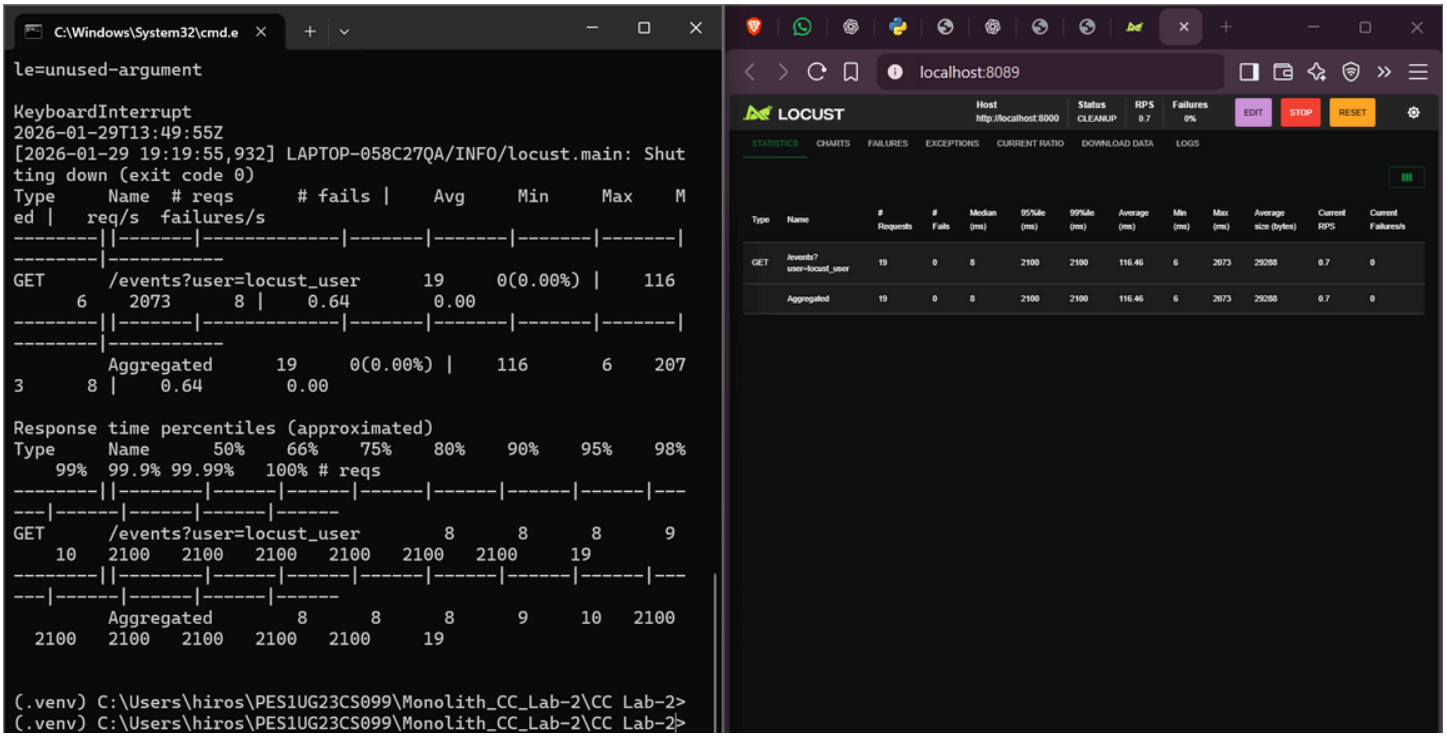


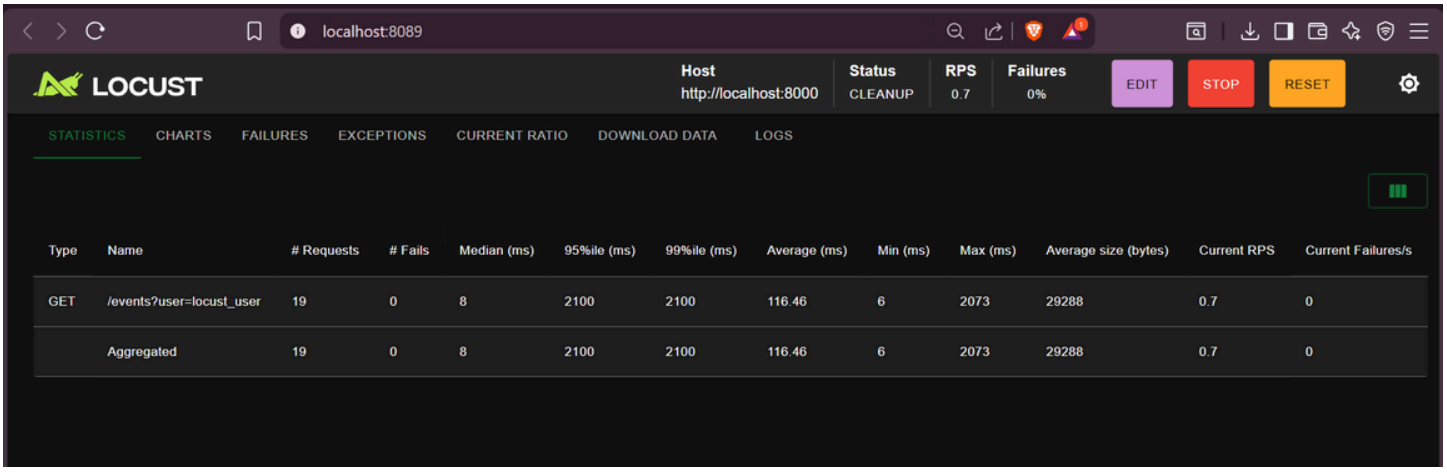
SS7:after events



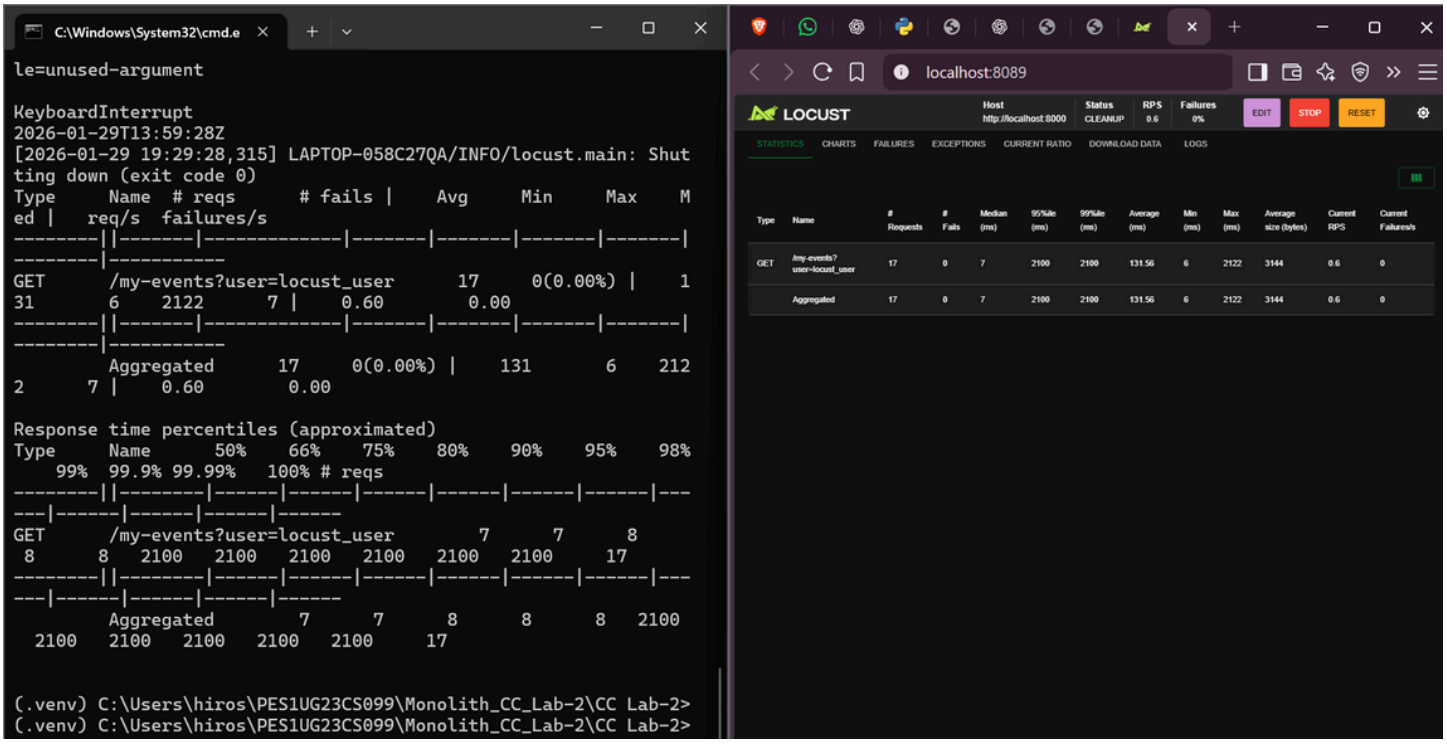


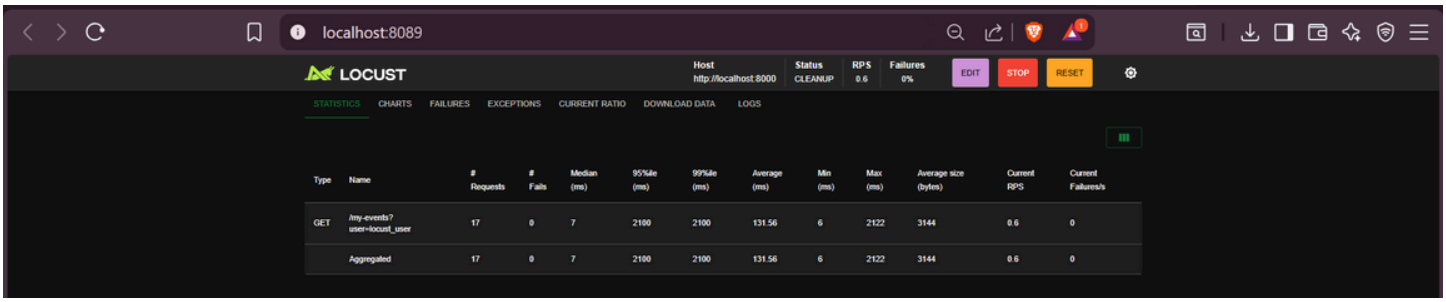
SS8: BEFORE MY EVENTS





SS9: AFTER MY EVENTS





Route-1:

What was the bottleneck?

The /events page had an unnecessary loop that kept running every time the page was loaded. This extra computation didn't serve any real purpose and slowed down the response.

What change did you make?

I removed the unnecessary loop and kept only the required database query and page rendering logic.

Why did the performance improve?

Since the extra computation was removed, the server had less work to do for each request, which reduced the response time and improved overall performance.

Route-2:

What was the bottleneck?

The /my-events route contained a loop that performed meaningless iterations on every request, which increased processing time.

What change did you make?

I removed the redundant loop so that the route only executes the database query and displays the results.

Why did the performance improve?

By eliminating unnecessary processing, the route became lighter and faster, resulting in better performance under load.

Short explanation of optimizations

For the /events route, an unnecessary CPU-intensive loop was present inside the route handler. This loop performed meaningless computations on every request. Removing this loop significantly reduced processing overhead and improved the route's performance under load.

Similarly, the /my-events route contained a dummy loop that wasted CPU cycles for each request. After removing this redundant computation, the route became lighter and responded faster.

Overall, these optimizations improved the performance of individual routes without changing the application's architecture. The application remains monolithic, as all components still run within a single codebase and deployment unit, but internal logic improvements led to better efficiency and responsiveness.