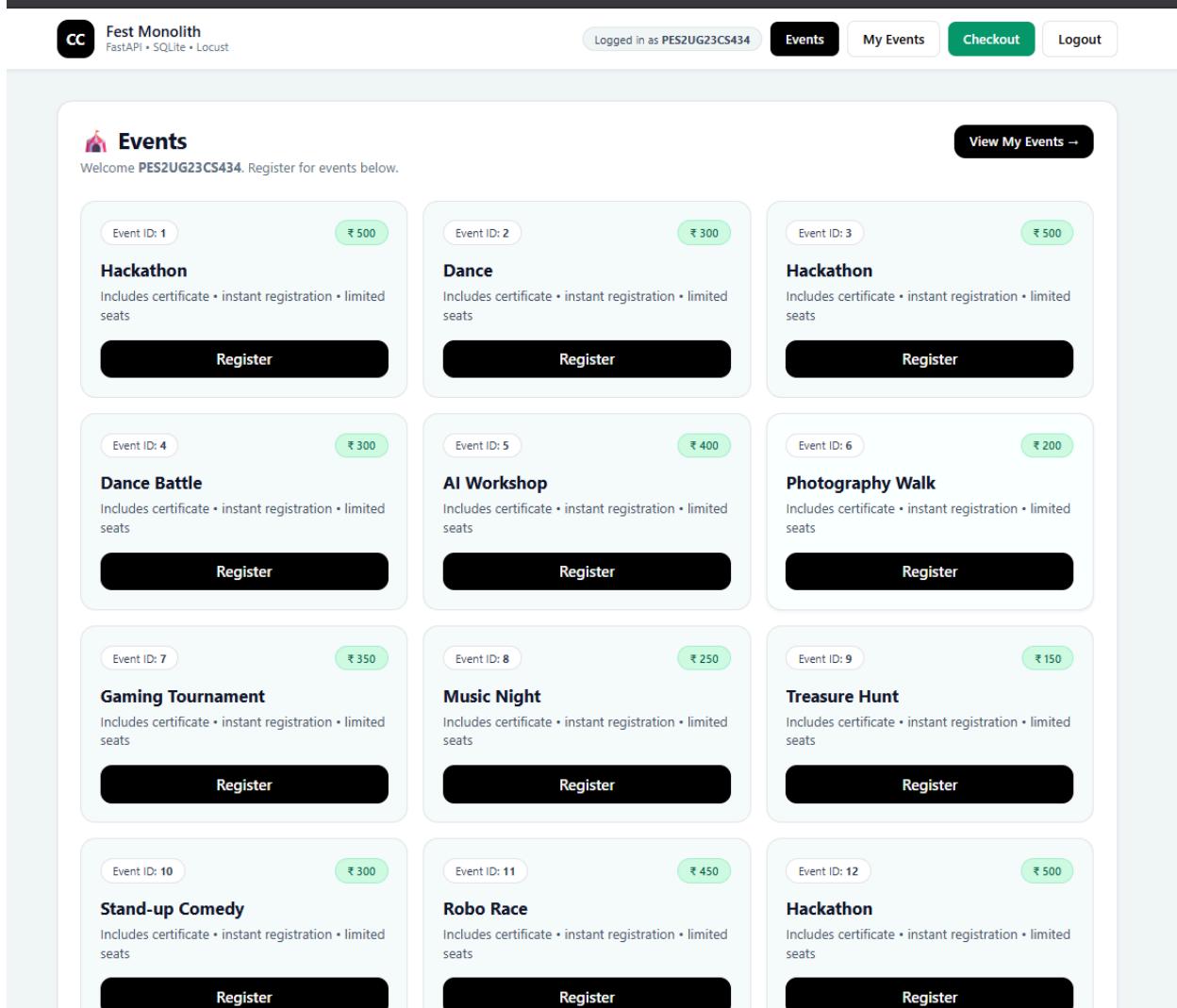


# CLOUD COMPUTING

## LAB 2

**NAME: PRANITH K**  
**SRN: PES2UG23CS434**  
**SECTION: G**

**SS1:**



The screenshot shows a web application interface for event registration. At the top, there is a header bar with the logo 'Fest Monolith' (CC), the text 'FastAPI + SQLite + Locust', and navigation links for 'Events', 'My Events', 'Checkout', and 'Logout'. The main content area is titled 'Events' and displays a grid of 12 event cards. Each card contains the event ID, name, price, a brief description, and a 'Register' button.

Event ID	Event Name	Price	Description	Action
1	Hackathon	₹ 500	Includes certificate • instant registration • limited seats	Register
2	Dance	₹ 300	Includes certificate • instant registration • limited seats	Register
3	Hackathon	₹ 500	Includes certificate • instant registration • limited seats	Register
4	Dance Battle	₹ 300	Includes certificate • instant registration • limited seats	Register
5	AI Workshop	₹ 400	Includes certificate • instant registration • limited seats	Register
6	Photography Walk	₹ 200	Includes certificate • instant registration • limited seats	Register
7	Gaming Tournament	₹ 350	Includes certificate • instant registration • limited seats	Register
8	Music Night	₹ 250	Includes certificate • instant registration • limited seats	Register
9	Treasure Hunt	₹ 150	Includes certificate • instant registration • limited seats	Register
10	Stand-up Comedy	₹ 300	Includes certificate • instant registration • limited seats	Register
11	Robo Race	₹ 450	Includes certificate • instant registration • limited seats	Register
12	Hackathon	₹ 500	Includes certificate • instant registration • limited seats	Register

## SS2:

### 💥 Monolith Failure

One bug in one module impacted the **entire application**.

HTTP 500

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

[Back to Events](#)

[Login](#)

```
~~^~~  
ZeroDivisionError: division by zero  
INFO:      127.0.0.1:62763 - "GET /checkout HTTP/1.1" 500 Internal Server Err  
or  
ERROR:     Exception in ASGT application
```

## SS3:

### Checkout

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

Total Payable

₹ 6600

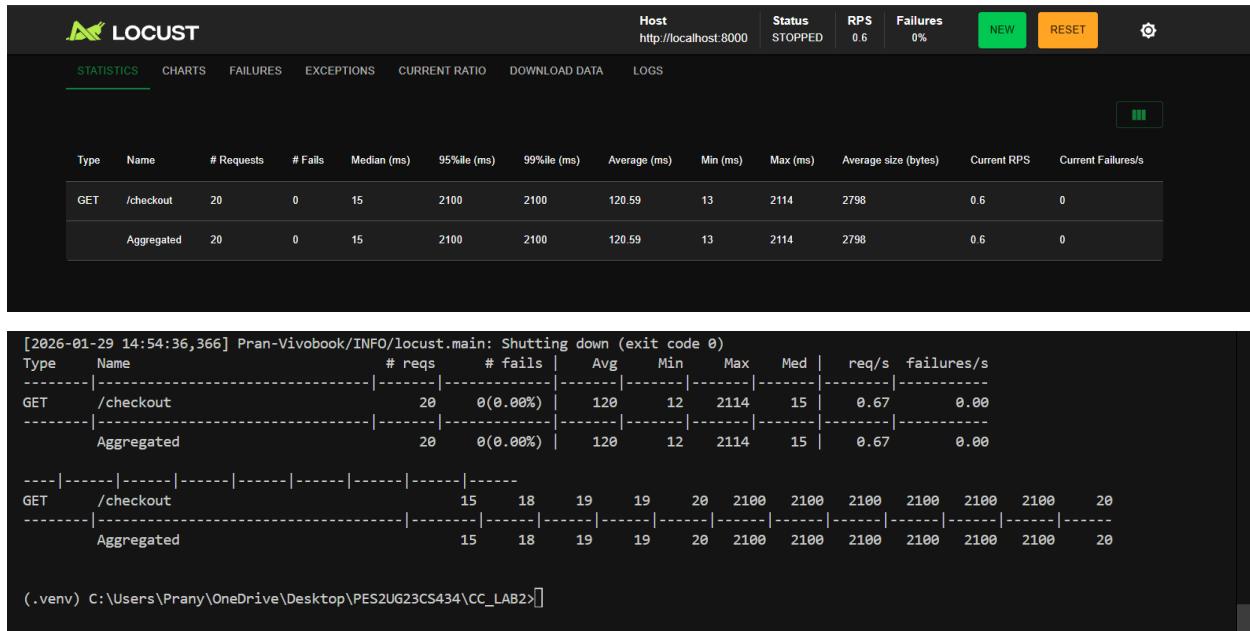
After fixing + optimizing checkout logic, re-run Locust and compare results.

#### 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).

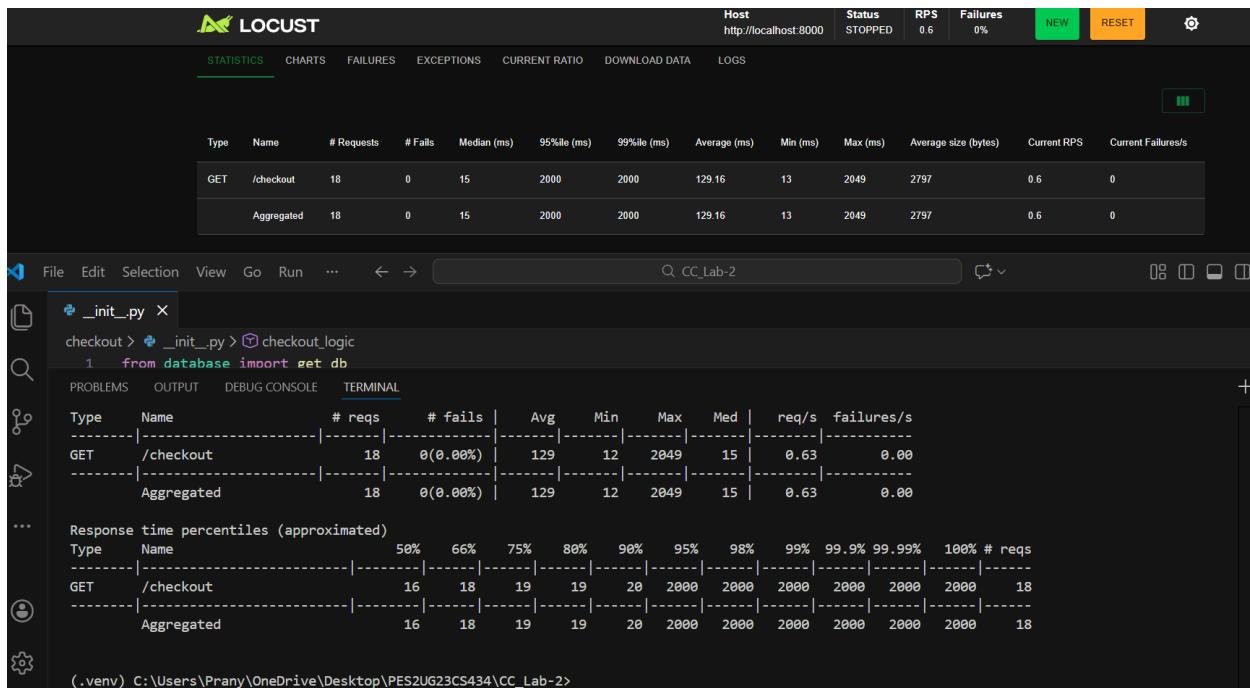
## SS4: PRE OPTIMIZATION



The screenshot shows the Locust web interface. At the top, it displays "Host http://localhost:8000", "Status STOPPED", "RPS 0.6", and "Failures 0%". Below this is a navigation bar with tabs: STATISTICS (selected), CHARTS, FAILURES, EXCEPTIONS, CURRENT RATIO, DOWNLOAD DATA, and LOGS. The STATISTICS tab shows a table with two rows: one for a specific request ('/checkout') and one for an aggregated view. The table includes columns for Type, Name, # Requests, # Fails, Median (ms), 95%ile (ms), 99%ile (ms), Average (ms), Min (ms), Max (ms), Average size (bytes), Current RPS, and Current Failures/s. The aggregated row shows identical values to the specific row. Below the table is a log window containing command-line output from a Python script named 'CC\_Lab2'. The log shows the script shutting down due to an exit code of 0.

```
[2026-01-29 14:54:36,366] Pran-Vivobook/INFO/locust.main: Shutting down (exit code 0)
Type      Name      # reqs    # fails | Avg   Min   Max   Med | req/s failures/s
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
GET      /checkout  20       0(0.00%) | 120   12    2114  15   | 0.67   0.00
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
Aggregated          20       0(0.00%) | 120   12    2114  15   | 0.67   0.00
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
GET      /checkout  15       18     19    19    20    2100  2100  2100  2100  2100  2100  20
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
Aggregated          15       18     19    19    20    2100  2100  2100  2100  2100  2100  20
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
(.venv) C:\Users\Prany\OneDrive\Desktop\PES2UG23CS434\CC_Lab2>[
```

## SS5: POST OPTIMIZATION - MINOR OPTIMIZATION OBSERVED



The screenshot shows the Locust web interface after optimization. The top bar remains the same. The STATISTICS tab is selected, showing a table with two rows: one for a specific request ('/checkout') and one for an aggregated view. The table includes columns for Type, Name, # Requests, # Fails, Median (ms), 95%ile (ms), 99%ile (ms), Average (ms), Min (ms), Max (ms), Average size (bytes), Current RPS, and Current Failures/s. The aggregated row shows identical values to the specific row. Below the table is a code editor window titled '\_init\_.py' with a single line of code: 'from database import set\_db'. The code editor has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. To the right of the code editor is a terminal window showing the command-line output from the 'CC\_Lab-2' script. The output shows the script shutting down due to an exit code of 0, with performance metrics like RPS and fail percentage being lower than in the pre-optimization phase.

```
checkout > _init_.py > checkout_logic
1   from database import set_db
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Type      Name      # reqs    # fails | Avg   Min   Max   Med | req/s failures/s
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
GET      /checkout  18       0(0.00%) | 129   12    2049  15   | 0.63   0.00
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
Aggregated          18       0(0.00%) | 129   12    2049  15   | 0.63   0.00
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
Response time percentiles (approximated)
Type      Name      50%   66%   75%   80%   90%   95%   98%   99%   99.9% 99.99% 100% # reqs
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
GET      /checkout  16     18     19     19     20    2000  2000  2000  2000  2000  2000  18
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
Aggregated          16     18     19     19     20    2000  2000  2000  2000  2000  2000  18
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
(.venv) C:\Users\Prany\OneDrive\Desktop\PES2UG23CS434\CC_Lab-2>
```

## SS6: EVENT OPTIMIZATION

The screenshot shows the Locust interface with the following details:

**Host:** http://localhost:8000 | **Status:** STOPPED | **RPS:** 0.6 | **Failures:** 0% | **NEW** | **RESET**

**STATISTICS** tab selected.

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	/events?user=locust_user	16	0	280	2500	2500	419.97	227	2480	21138	0.6	0
Aggregated		16	0	280	2500	2500	419.97	227	2480	21138	0.6	0

**TERMINAL** tab selected.

```
checkout > __init__.py > checkout_logic
  1   from database import set_db
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

Response time percentiles (approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%	99.99%
% 100% # reqs											
GET	/events?user=locust_user	280	300	360	360	400	2500	2500	2500	2500	250
0	2500 16										
Aggregated		280	300	360	360	400	2500	2500	2500	2500	250
0	2500 16										

(.venv) C:\Users\Prany\OneDrive\Desktop\PES2UG23CS434\CC\_Lab-2>

## SS7: OPTIMIZED EVENTS

The screenshot shows the Locust interface with the following details:

**Host:** http://localhost:8000 | **Status:** STOPPED | **RPS:** 0.6 | **Failures:** 0% | **NEW** | **RESET**

**STATISTICS** tab selected.

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	/events?user=locust_user	19	0	19	2100	2100	126.34	14	2090	21138	0.6	0
Aggregated		19	0	19	2100	2100	126.34	14	2090	21138	0.6	0

**TERMINAL** tab selected.

```
Selection  View  Go  Run  ...  <  >  Q CC_Lab-2  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

main.py checkout

```
__init__.py
main.py > events
58
```

Response time percentiles (approximated)

Type	Name	50%	66%	75%	80%	90%	95%	98%	99%	99.9%	99.99%
% 100% # reqs											
GET	/events?user=locust_user	19	19	20	20	25	2100	2100	2100	2100	2100
0	2100 19										
Aggregated		19	19	20	20	25	2100	2100	2100	2100	2100
0	2100 19										

(.venv) C:\Users\Prany\OneDrive\Desktop\PES2UG23CS434\CC\_Lab-2>

## SS8: PRE OPTIMIZATION

The screenshot shows the Locust performance testing interface. At the top, there's a navigation bar with tabs: STATISTICS (highlighted in green), CHARTS, FAILURES, EXCEPTIONS, CURRENT RATIO, DOWNLOAD DATA, and LOGS. Below the navigation bar is a table with performance metrics. The table has columns for Type, Name, # Requests, # Fails, Median (ms), 95%ile (ms), 99%ile (ms), Average (ms), Min (ms), Max (ms), Average size (bytes), Current RPS, and Current Failures. Two rows are present: one for a specific request ('GET /my-events?user=locust\_user') and one for an aggregated summary. The bottom half of the screen displays a terminal window showing detailed performance data and response time percentiles. The terminal output includes sections for 'Response time percentiles (approximated)' and a table of statistics for the same two rows as the table above.

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
GET	/my-events?user=locust_user	17	0	100	2200	2200	228.47	96	2177	3144	0.6	0
	Aggregated	17	0	100	2200	2200	228.47	96	2177	3144	0.6	0

File Edit Selection View Go Run ... ← → 🔍 CC\_Lab-2 ⚡

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
GET      /my-events?user=locust_user    17    0(0.00%) | 228    96   2176   100 | 0.59     0.00
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
      Aggregated          17    0(0.00%) | 228    96   2176   100 | 0.59     0.00

Response time percentiles (approximated)
Type      Name      50%  66%  75%  80%  90%  95%  98%  99%  99.9% 99.99% 100% # reqs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
GET      /my-events?user=locust_user  100  110  110  120  160  2200  2200  2200  2200  2200  2200  17
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
      Aggregated          100  110  110  120  160  2200  2200  2200  2200  2200  2200  17
```

## SS9: POST OPTIMIZATION

## **QNA:**

### **Short Explanation for Both Routes**

**Route: /events**

#### **What was the bottleneck?**

The events route had extra processing logic that was executed on every request, which increased the response time.

#### **What change did you make?**

I removed the unnecessary processing while keeping the database access and page output the same.

#### **Why did the performance improve?**

By reducing the amount of work done per request, the server was able to handle the request more efficiently.

**Route: /my-events**

#### **What was the bottleneck?**

The route included redundant operations that caused additional CPU usage during each request.

#### **What change did you make?**

I removed the redundant code without changing the functionality of the route.

#### **Why did the performance improve?**

Since fewer computations were performed, the response time improved during load testing.