

CC Lab 3

Name : Vedant Varma

SRN : PES1UG22CS681

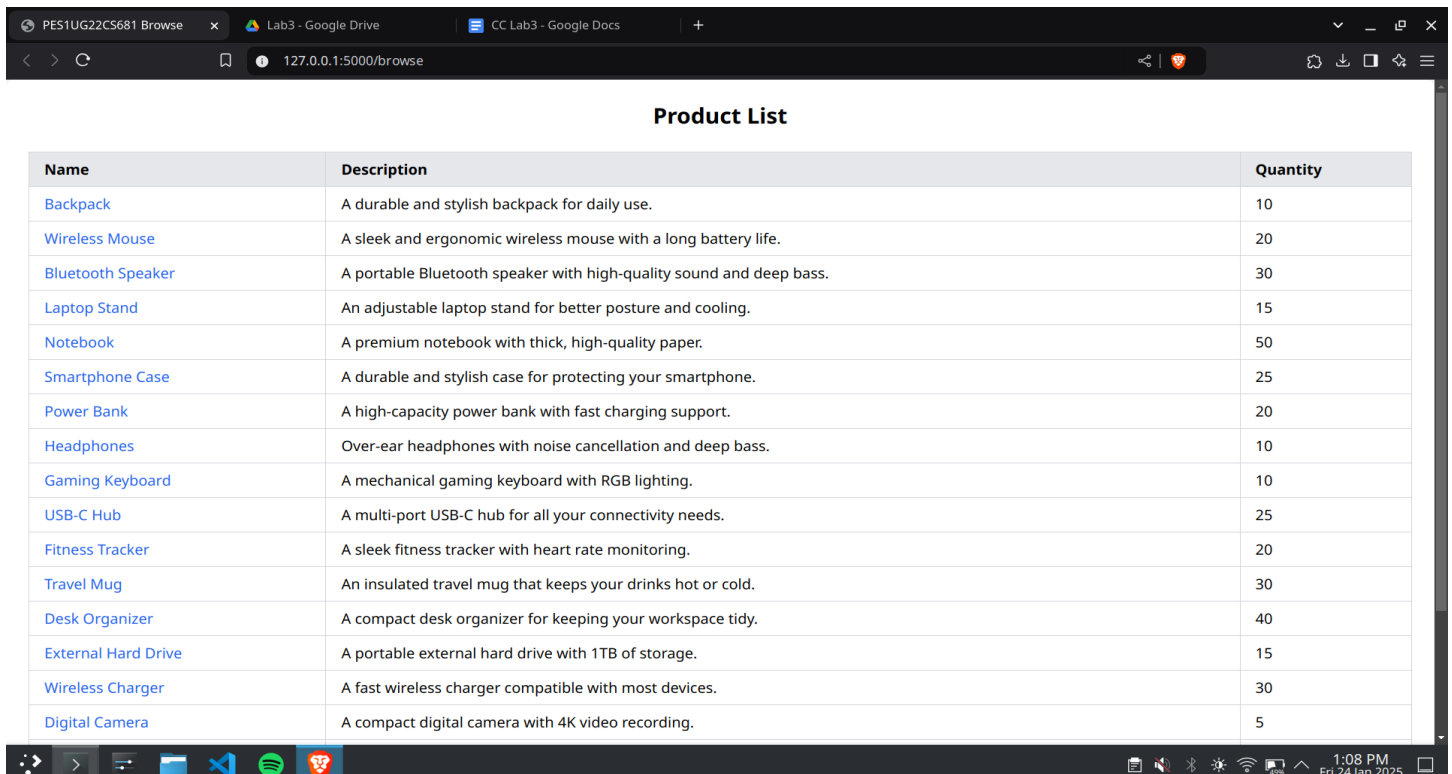
Section : L section 6th Sem CSE

SS1 :

=> SRN has been modified and it is visible in the title of the website

=> We finished the registration with the username as Vedant Varma and the password has been set.

=> Upon logging in we get redirected to the browse page.



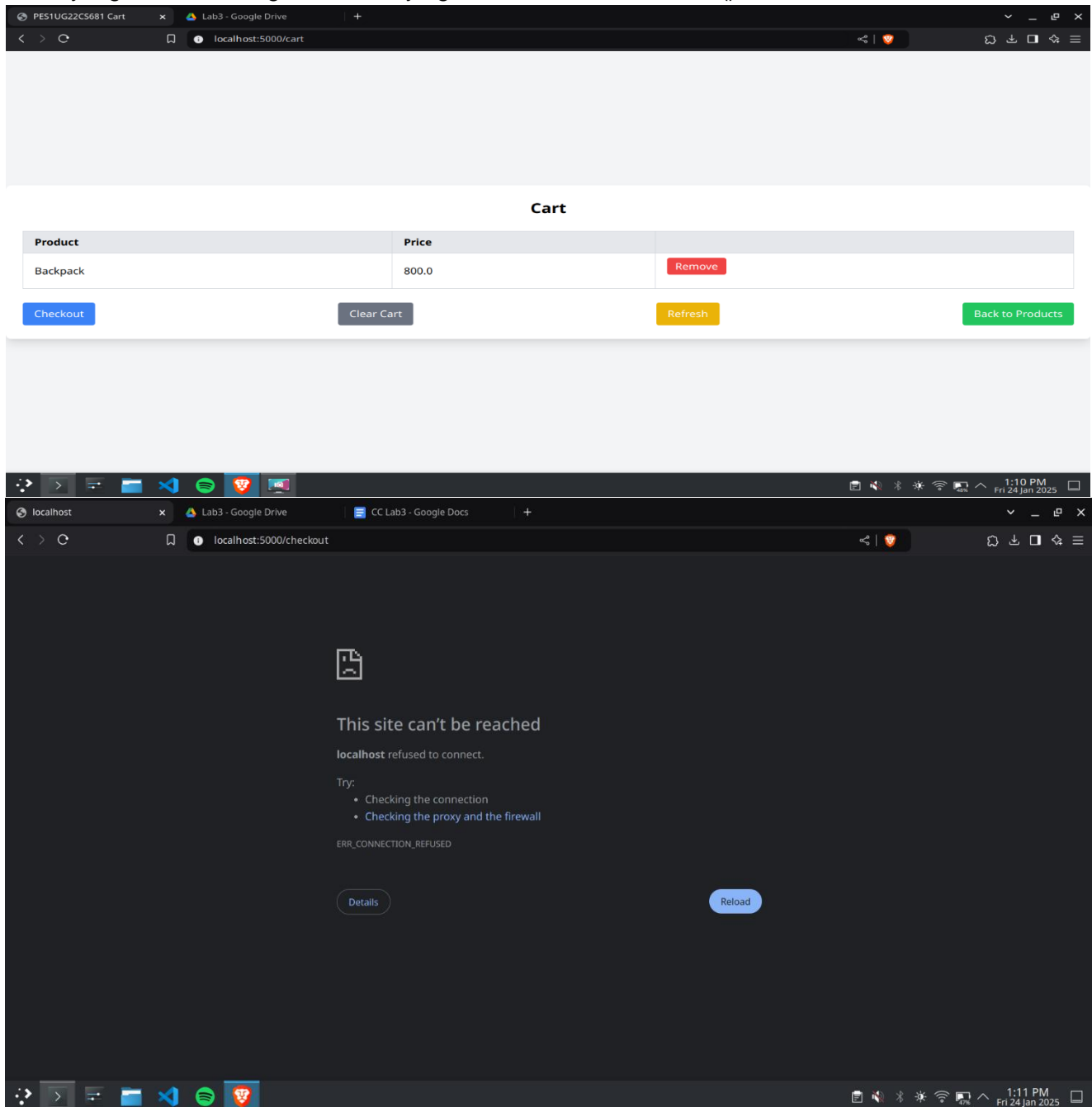
Name	Description	Quantity
Backpack	A durable and stylish backpack for daily use.	10
Wireless Mouse	A sleek and ergonomic wireless mouse with a long battery life.	20
Bluetooth Speaker	A portable Bluetooth speaker with high-quality sound and deep bass.	30
Laptop Stand	An adjustable laptop stand for better posture and cooling.	15
Notebook	A premium notebook with thick, high-quality paper.	50
Smartphone Case	A durable and stylish case for protecting your smartphone.	25
Power Bank	A high-capacity power bank with fast charging support.	20
Headphones	Over-ear headphones with noise cancellation and deep bass.	10
Gaming Keyboard	A mechanical gaming keyboard with RGB lighting.	10
USB-C Hub	A multi-port USB-C hub for all your connectivity needs.	25
Fitness Tracker	A sleek fitness tracker with heart rate monitoring.	20
Travel Mug	An insulated travel mug that keeps your drinks hot or cold.	30
Desk Organizer	A compact desk organizer for keeping your workspace tidy.	40
External Hard Drive	A portable external hard drive with 1TB of storage.	15
Wireless Charger	A fast wireless charger compatible with most devices.	30
Digital Camera	A compact digital camera with 4K video recording.	5

SS2 :

=> On adding items like a backpack as seen in the screenshot below, we get redirected to the cart page and on clicking checkout we get an error as mentioned in the docs.

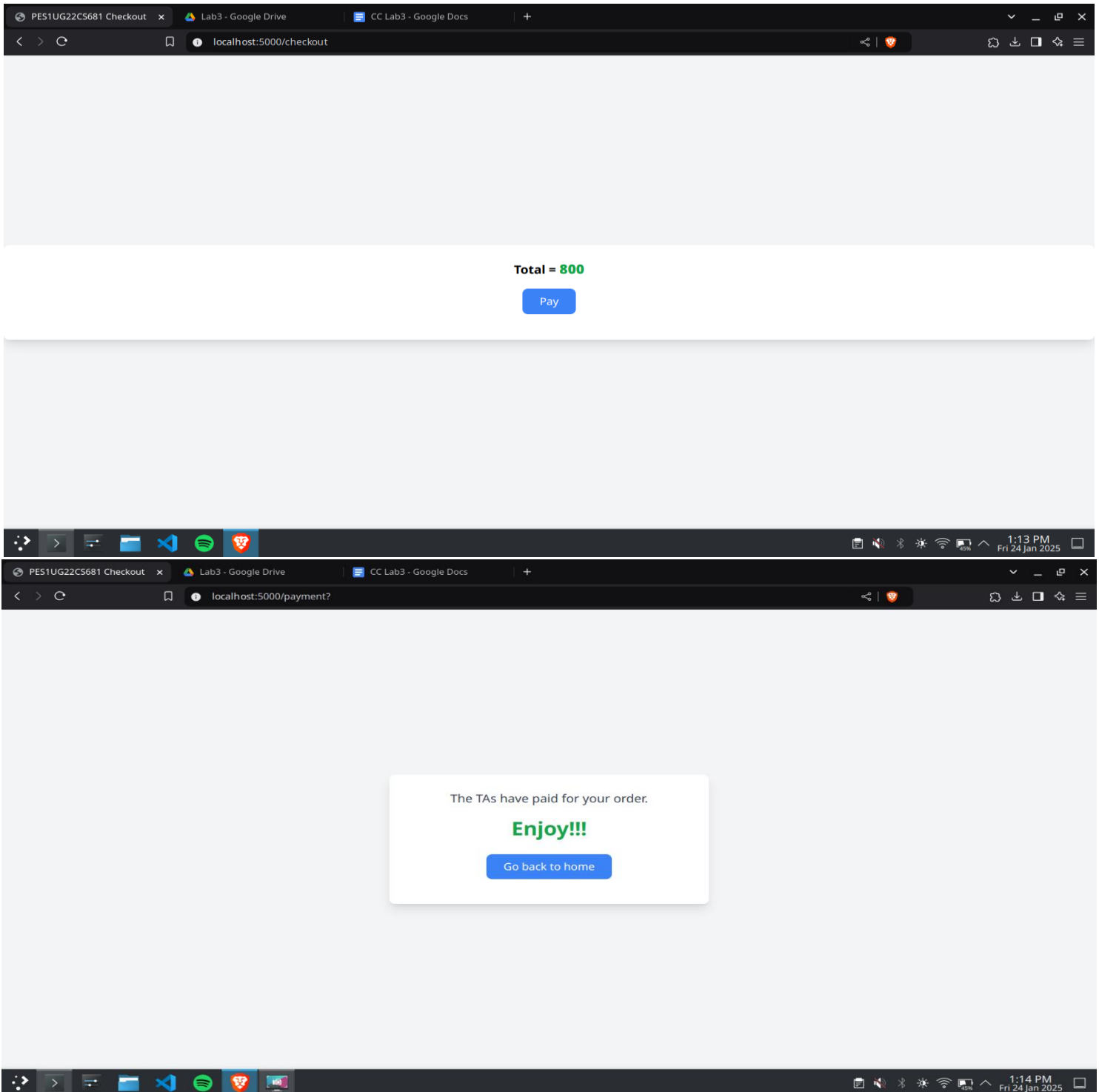
=> The reason we get this error is because `os._exit(1)` will stop the execution before returning total.

=> Any logic in the calling function relying on the result of `checkout()` will break.



SS3 :

=> We fixed the `__init__.py` in the checkout route by commenting the `os._exit(1)` line



SS4 :

=> Here we use only 1 user and 1 ramp up that is tested for 30s.

=> As we can see the average time is 26.7ms for 1121 requests

The screenshot shows the Locust web interface on the left and a terminal window on the right. The Locust interface displays statistics for a test named '/checkout' with 1121 requests, 0 failures, and an average response time of 26.7ms. The terminal window shows the execution of the Locust test, including the start time, the number of users spawned, and the final statistics.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)
GET	/checkout	1121	0	26	33	35	26.7	23
Aggregated		1121	0	26	33	35	26.7	23

```
(cloud_python) vedant@vedant: ~/Documents/PES/6th_Sem/CC/Labs/PES1UG22CS681/CC_Monolith
locust -f ./CC_Monolith/locust/checkout-locustfile.py
[2025-01-27 09:54:26,590] vedant/INFO/locust.main: Starting Locust 2.32.6
[2025-01-27 09:54:26,590] vedant/INFO/locust.main: Starting web interface at http://0.0.0.0:8089
[2025-01-27 09:54:38,964] vedant/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 p
er second
[2025-01-27 09:54:38,974] vedant/INFO/locust.runners: All users spawned: {"checkout": 1} (1
total users)
KeyboardInterrupt
[2025-01-27 09:55:28,640] vedant/INFO/locust.main: Shutting down (exit code 0)
Type Name # reqs # fails Avg Min Max Med req/s failures/s
GET /checkout 1121 0(0.00%) 26 22 41 26 37.38 0.0
Aggregated 1121 0(0.00%) 26 22 41 26 37.38 0.0
Response time percentiles (approximated)
Type Name # reqs 50% 66% 75% 80% 90% 95% 98% 99% 99.9% 99.99
GET /checkout 1121 26 27 28 31 33 35 39 4
Aggregated 1121 26 27 28 31 33 35 39 4
```

SS5 :

=> On replacing the code and optimizing it we can see that for the similar users (1) and ramp up (1) values for 30s, this gives us an average time of 12.37ms for 2414 requests.

The screenshot shows the Locust web interface on the left and a terminal window on the right. The Locust interface displays statistics for a test named '/checkout' with 2414 requests, 0 failures, and an average response time of 12.37ms. The terminal window shows the execution of the Locust test, including the start time, the number of users spawned, and the final statistics.

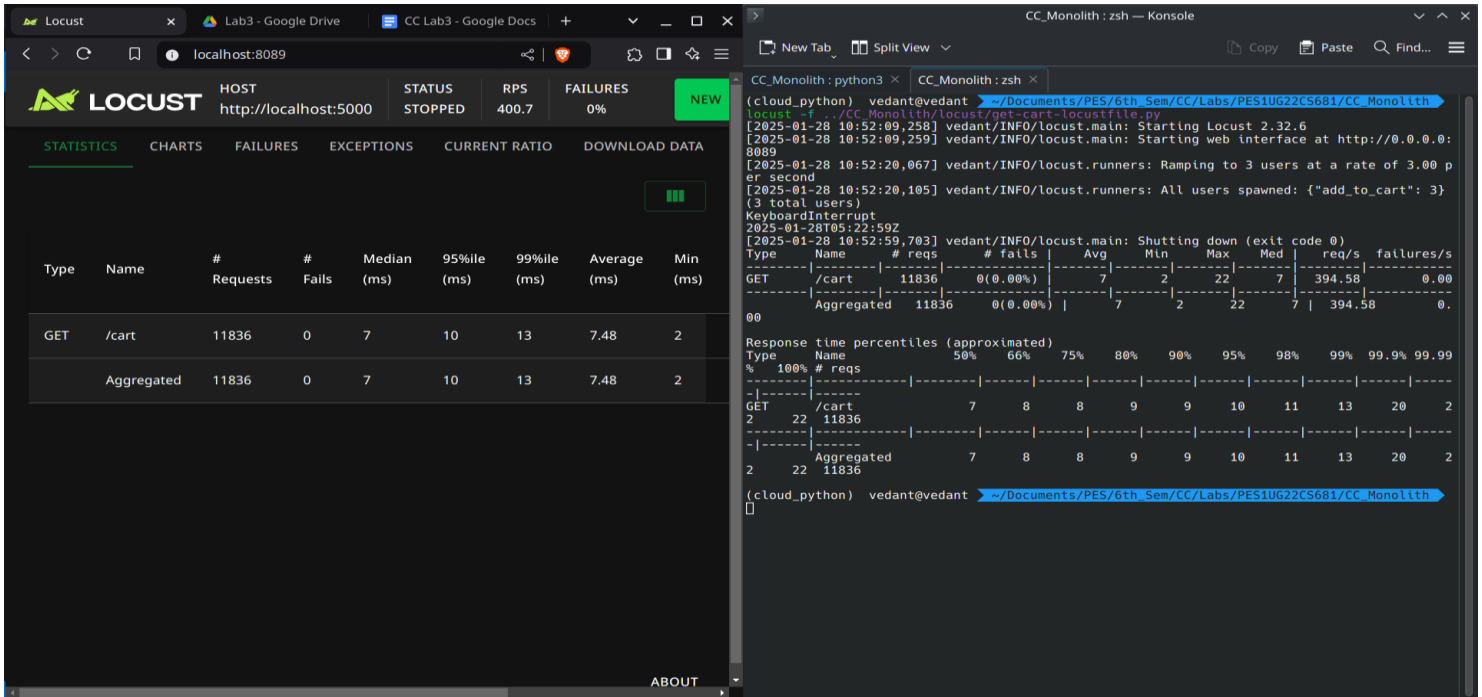
Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)
GET	/checkout	2414	0	12	17	21	12.37	6
Aggregated		2414	0	12	17	21	12.37	6

```
(cloud_python) vedant@vedant: ~/Documents/PES/6th_Sem/CC/Labs/PES1UG22CS681/CC_Monolith
locust -f ./CC_Monolith/locust/checkout-locustfile.py
[2025-01-27 09:59:32,241] vedant/INFO/locust.main: Starting Locust 2.32.6
[2025-01-27 09:59:32,241] vedant/INFO/locust.main: Starting web interface at http://0.0.0.0:8089
[2025-01-27 09:59:46,204] vedant/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 p
er second
[2025-01-27 09:59:46,219] vedant/INFO/locust.runners: All users spawned: {"checkout": 1} (1
total users)
KeyboardInterrupt
[2025-01-27 10:00:25,693] vedant/INFO/locust.main: Shutting down (exit code 0)
Type Name # reqs # fails Avg Min Max Med req/s failures/s
GET /checkout 2414 0(0.00%) 12 6 61 12 80.44 0.0
Aggregated 2414 0(0.00%) 12 6 61 12 80.44 0.0
Response time percentiles (approximated)
Type Name # reqs 50% 66% 75% 80% 90% 95% 98% 99% 99.9% 99.99
GET /checkout 2414 12 13 14 15 16 17 20 21 49 6
Aggregated 2414 12 13 14 15 16 17 20 21 49 6
```

SS6 :

=> This is the screenshot of the unoptimized route for /cart. We use 3 users with the ramp up value as 3 for 30s.

=> As we can see the average time is 7.48ms for 11836 requests



SS7 :

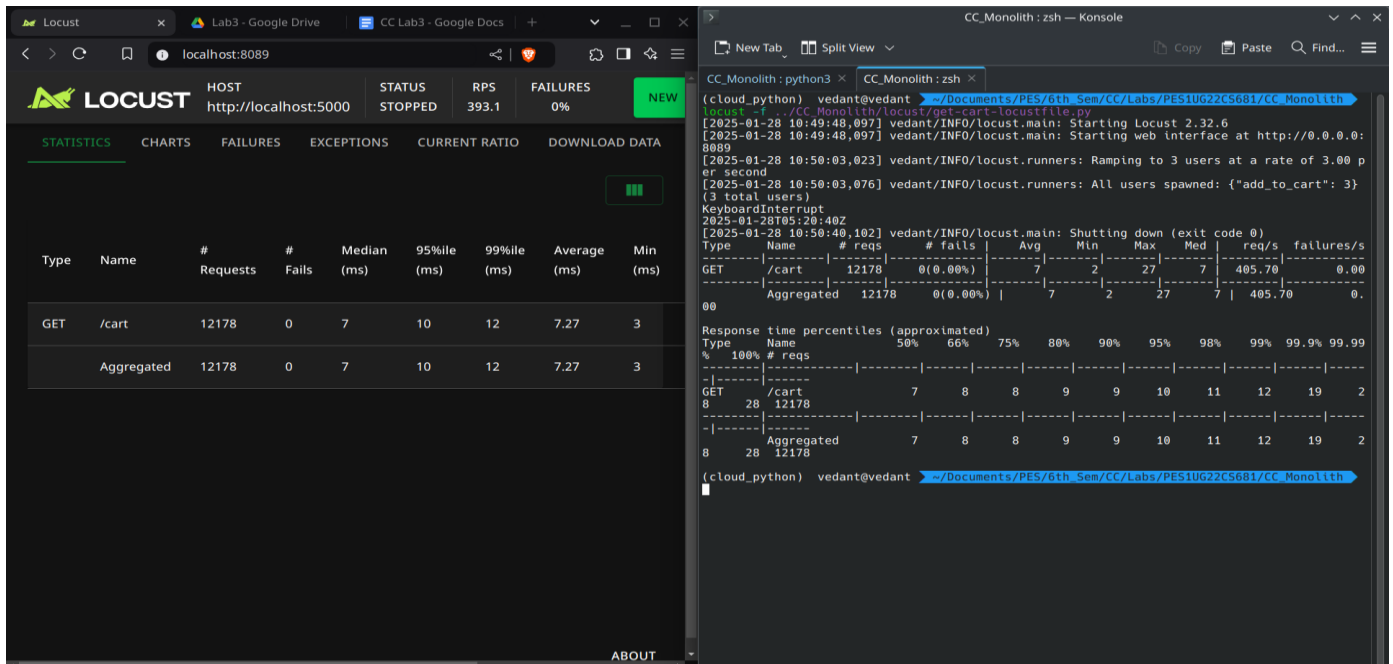
=> We need to optimize the /cart route. In the __init__.py file under the /cart route we can see that evals and the for loops are used too much.

=> We replace these with json.loads and list comprehensions for the following reasons :

- 1) eval is slower because it processes the string as Python code, invoking the interpreter and handling more overhead whereas json.loads is optimized for parsing JSON and is faster in comparison for JSON-specific tasks
- 2) List comprehensions are implemented in C at the interpreter level, making them more efficient than the equivalent for loop with list.append(), which involves additional overhead from Python's function calls

Hence we replace and optimize the code and write the function as follows :

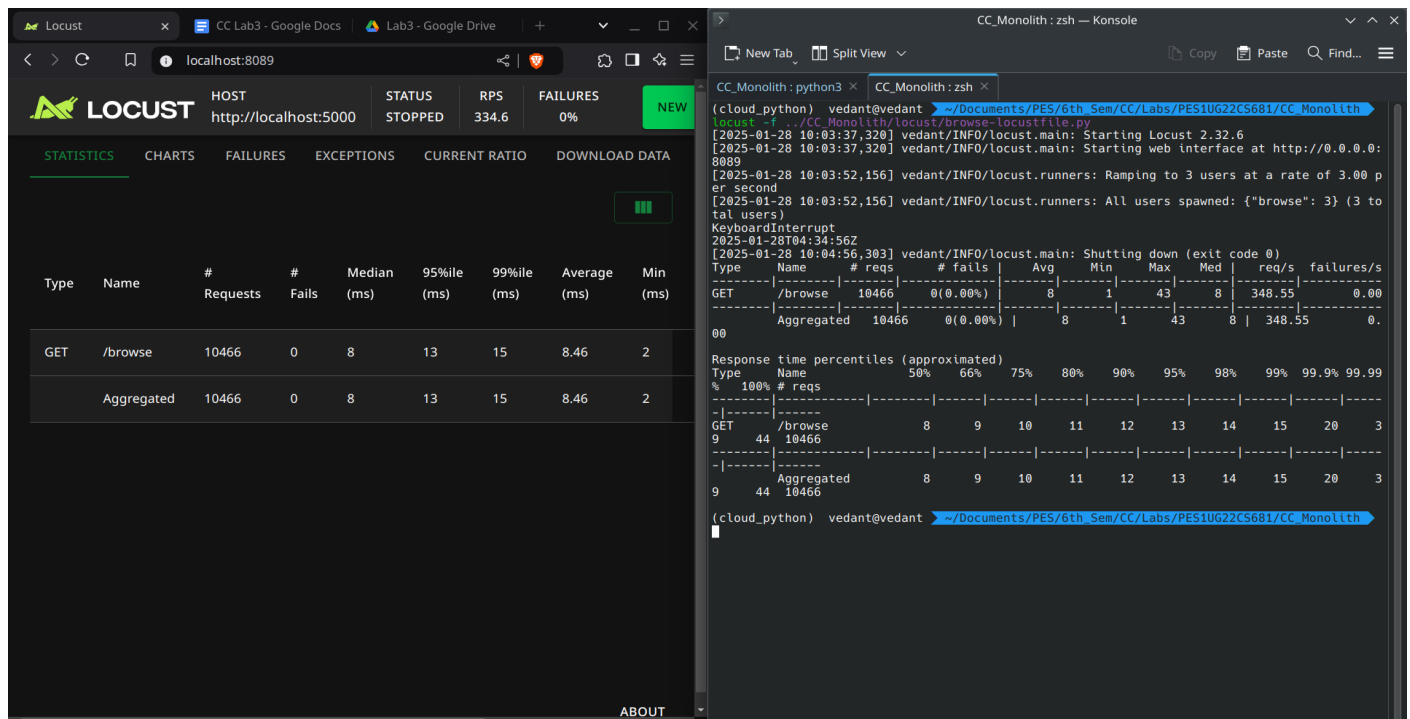
```
def get_cart(username: str) -> list:
    cart_details = dao.get_cart(username)
    if cart_details is None:
        return []
    ids = [j for i in cart_details
           for j in loads(i['contents'])]
    return [products.get_product(i) for i in ids]
```



We can see that the /cart route has been optimized and the average time has been reduced to 7.27ms for the same number of users(3) and ramp up (3) value tested for 30s

SS8 :

As done for /cart, we perform a similar comparison for /browse. We use 3 users and the ramp up value is 3 which is tested for 30s. The average time is 8.46ms for 10466 requests



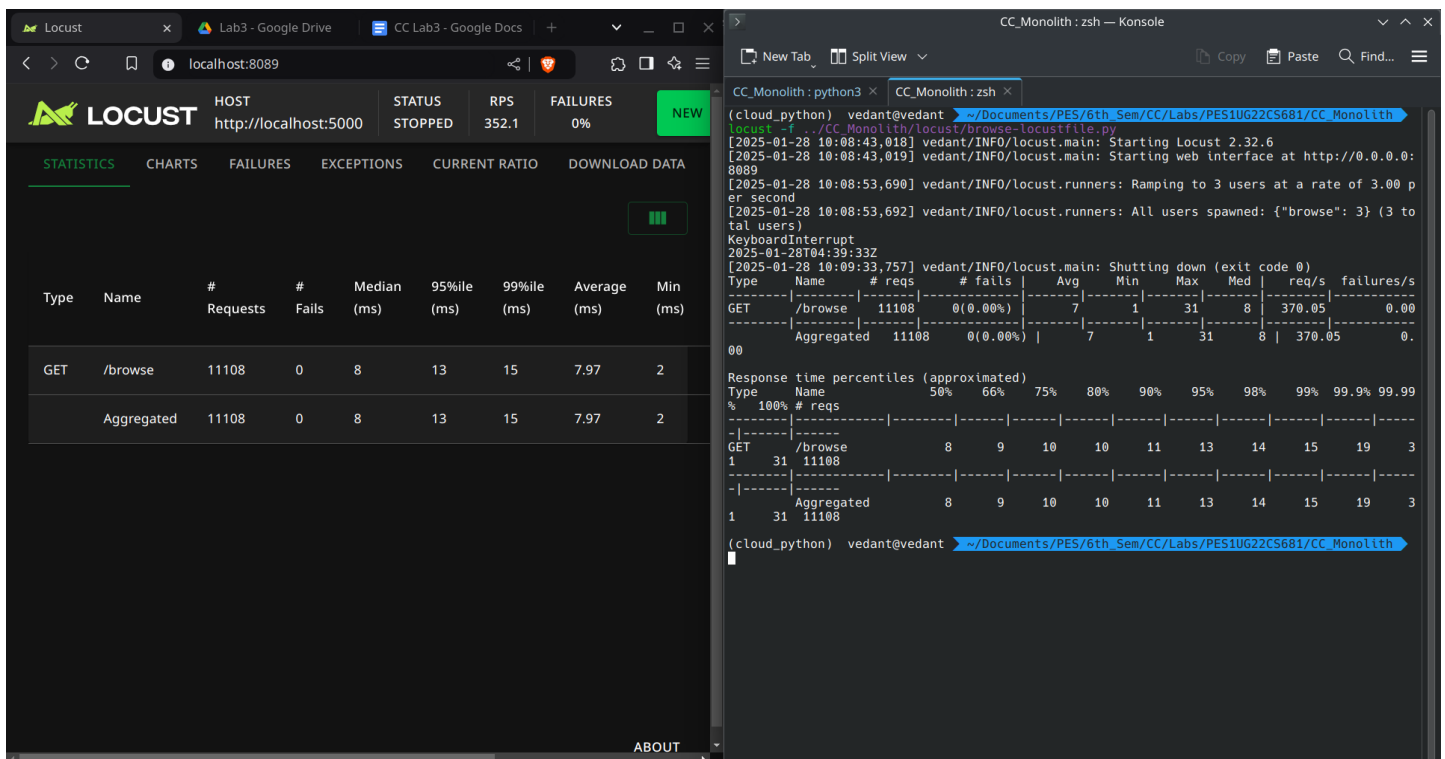
SS9 :

=> We need to optimize the /product route. In the __init__.py file under the /product route we can see that the for loops can be replaced with list comprehensions like done in the /cart route

=> List comprehensions are implemented in C at the interpreter level, making them more efficient than the equivalent for loop with list.append(), which involves additional overhead from Python's function calls

=> Hence in the method list_products defined in the __init__.py present in the /product route we replace the code by using a list comprehension :

```
def list_products() -> list[Product]:  
    result = [Product.load(data) for data in dao.list_products()]  
    return result
```



As mentioned earlier we use 3 users and ramp up as 3 and it's evident that the average time has been reduced to 7.97ms for 11108 requests