

CLOUD COMPUTING LAB 3

Name: CHAITRA V
SEM & SECTION: 6 'A'

SRN: PES1UG22AM045
Date: 28/01/2025

Monolithic Applications: The Great Indian Paratha

Imagine walking into a dhaba, and the cook serves you a massive paratha stuffed with aloo, paneer, gobhi, and even some leftover dal from last night. That's your monolithic application—one big, all-inclusive piece of software. It has everything: the dough (UI), the stuffing (business logic), and the ghee on top (data layer).

At first glance, it looks great—one wholesome package, easy to eat, and no extra plates. But when it's time to break it apart or make changes (like adding some green chilies), you realize things aren't as simple as they seemed.

Advantages of Monolithic Applications: Why We Love the Paratha

1. Simple and Straightforward

Building a monolith is like rolling out one large paratha. All the ingredients go into a single dough (codebase), and it's cooked together. Easy to prepare, serve, and consume without fuss. Great for beginners and small teams!

2. Efficient Communication

In a paratha, the flavors of aloo and masalas blend beautifully because everything is together. Similarly, in a monolithic app, all parts (UI, business logic, and data) are tightly integrated, allowing them to work efficiently without the overhead of connecting through APIs or network calls.

3. Easier Debugging

If your paratha tastes off, you don't have to guess—everything is in one place. The same goes for monoliths. Troubleshooting is simpler since all components live within the same codebase.

4. Perfect for Small Kitchens (Teams)

A single cook (developer or small team) can handle a monolith. It's ideal when resources are limited, and you want to deliver something quickly.

Disadvantages of Monolithic Applications: When the Paratha Gets Too Big

1. Scaling Struggles

Imagine trying to roll out a paratha that's too big for the tawa. A monolithic app grows the same way—when the application gets larger, adding or updating features becomes increasingly difficult. Changing one thing might affect everything else.

2. Deployment Drama

Adding a pinch of ajwain (a new feature) to your paratha? You'll need to knead the dough and cook the entire thing all over again. Similarly, even small updates in a monolith require redeploying the entire application.

3. **Not Flexible with “Custom Orders”**

Let’s say someone only wants plain roti or just paneer stuffing. Too bad—a monolith can’t separate components. It’s all or nothing.

4. **Maintenance Challenges**

Imagine 20 cooks working on the same paratha, each with their own ideas about how much masala to add. Chaos! With a growing team and codebase, managing a monolith becomes messy and error-prone.

5. **Single Point of Failure**

Burn the paratha on one side, and the entire thing is ruined. Similarly, if one part of a monolith fails, the whole application can go down with it.

The Verdict: A Paratha or a Thali?

Monolithic applications are like a hearty paratha—simple, satisfying, and perfect for small setups. But as your needs grow, and your customers (users) demand variety, you may want to switch to a thali (microservices)—where each dish (service) is prepared and served separately. This way, scaling, maintenance, and customization become easier.

So, the choice is yours: stick to the comfort of a single paratha or upgrade to a thali to meet the demands of a bigger crowd!

Well, That Was Theory—Let’s Do a Practical!

To better understand a monolithic application, let’s dive into a small practical example.

We’ll provide a small **Flask app** that uses **HTML** for front-end and **SQLite3** as the database.

Why Is This a Monolithic Application?

In this setup:

- **All services (UI, business logic, and database) run within a single process.**
- The entire application is tightly coupled and works as a single cohesive unit.

The code for this practical will be provided as part of the lab.

(PS: The code probably has lots of bugs and ui is bare minimum, so bear with it please, or you could improve it and help us :))

VERY IMPORTANT!!!!

- If you're using Windows, please use a Linux Virtual Machine or WSL (Windows Subsystem for Linux) for the lab. Attempting to use native Windows may lead to a very horrible experience. You have been warned.
- Make sure your python version is ≥ 3.9 .
- Install locust only using pip!! (and NOT using apt).
- If you are on a mac disable AirPlay Receiver by searching for “Airplay Reciever” in spotlight search (Cmd+Spacebar) before starting.
- In WSL, make sure your project folder is under /home/<username>/<SRN> or /root/SRN and NOT under /mnt. You can do this by scrolling down on the left side of file explorer in windows till u find “linux”, then click on linux and then click on Ubuntu folder. Then navigate to home/<username> or root
- Git Bash is NOT WSL.

1. First create a directory with your SRN. Unzip the file provided to you in that folder.

```
mkdir PES1UG2XCSXXX
```

Once that's done the pwd output should look something like

```
/root/PES1UG2XCSXXX/CC_Monolith
```

2. Now we will run the code
 - a. So we will setup python virtual environment so that there is no collision between module versions. (NOTE: if pyenv doesnt exist u might need to apt install the python pyenv package)

```
python3 -m venv .venv  
source .venv/bin/activate
```

- b. Install the modules

```
pip install -r requirements.txt
```

3. We will start the server

```
python3 main.py
```

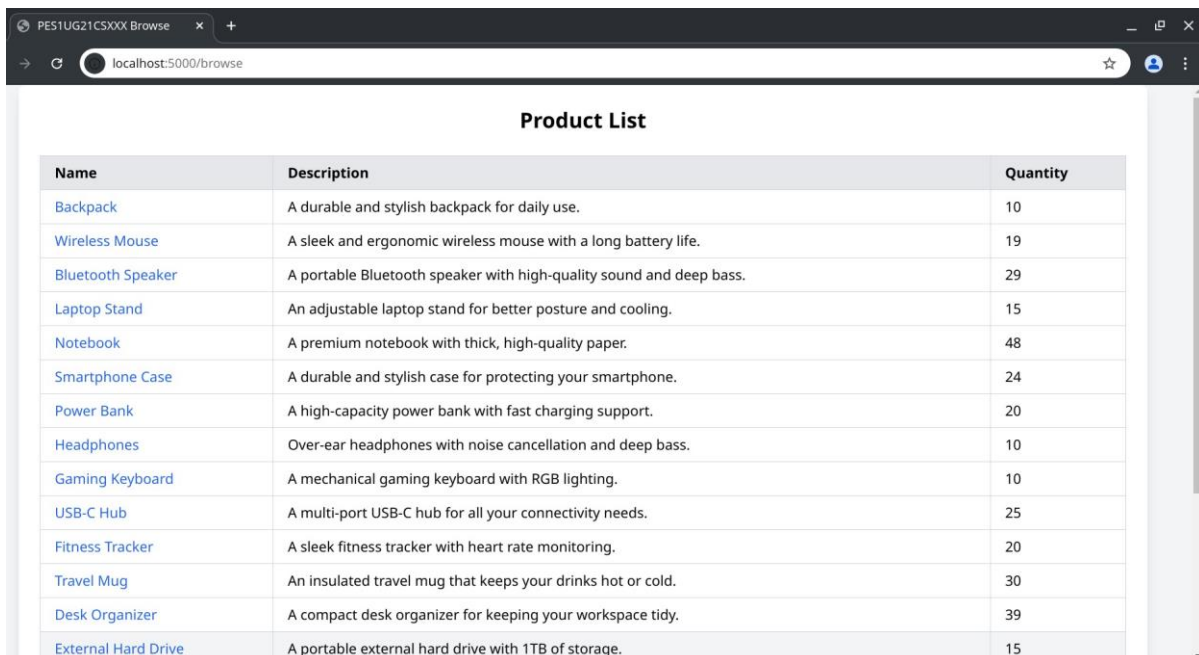
4. In *main.py* you have to **update the SRN**, on **line number 15**

```

1 import json
2
3 import flask
4 import jwt
5 from flask import render_template, request, redirect, url_for
6 import products
7 from auth import do_login, sign_up
8 from products import list_products
9 from cart import add_to_cart as ac, get_cart, remove_from_cart, delete_cart
10 from checkout import checkout as chk, complete_checkout
11
12 app = flask.Flask(__name__)
13 app.template_folder = 'templates'
14 SRN = "PES1UG21CSXXX"
15
16 @app.route('/')
17 def index():
18     return redirect(url_for('browse'))
19
20
21 @app.route('/cart')
22 def cart():
23     token = request.cookies.get('token')
24     if token is None:
25         return redirect(url_for('login'))
26     dec = jwt.decode(token, 'secret', algorithms=['HS256'])
27     username = dec['sub']
28     cart = get_cart(username)
29     return render_template('cart.jinja', cart=cart, srn=SRN)
30
31 @app.route('/cart/remove<id>', methods=['POST'])
32 def remove_cart_item(id):
33     token = request.cookies.get('token')
34     if token is None:

```

5. Visit the site (note if localhost didn't work try with 127.0.0.1)
 - a. <http://localhost:5000/register> to register a user
 - b. <http://localhost:5000/login> to login
 - c. <http://localhost:5000/browse> to browse for stuff, click on the product name to view the details about the product, and if you like the product go ahead and add the product to your cart. On successful setup and run the following screen should appear. This will be the **First Screenshot(SS1)**. Make sure your **SRN** is clearly visible.



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.	19
Bluetooth Speaker	A portable Bluetooth speaker with high-quality sound and deep bass.	29
Laptop Stand	An adjustable laptop stand for better posture and cooling.	15
Notebook	A premium notebook with thick, high-quality paper.	48
Smartphone Case	A durable and stylish case for protecting your smartphone.	24
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.	39
External Hard Drive	A portable external hard drive with 1TB of storage.	15

MY SCREENSHOT (SS1):

PTO

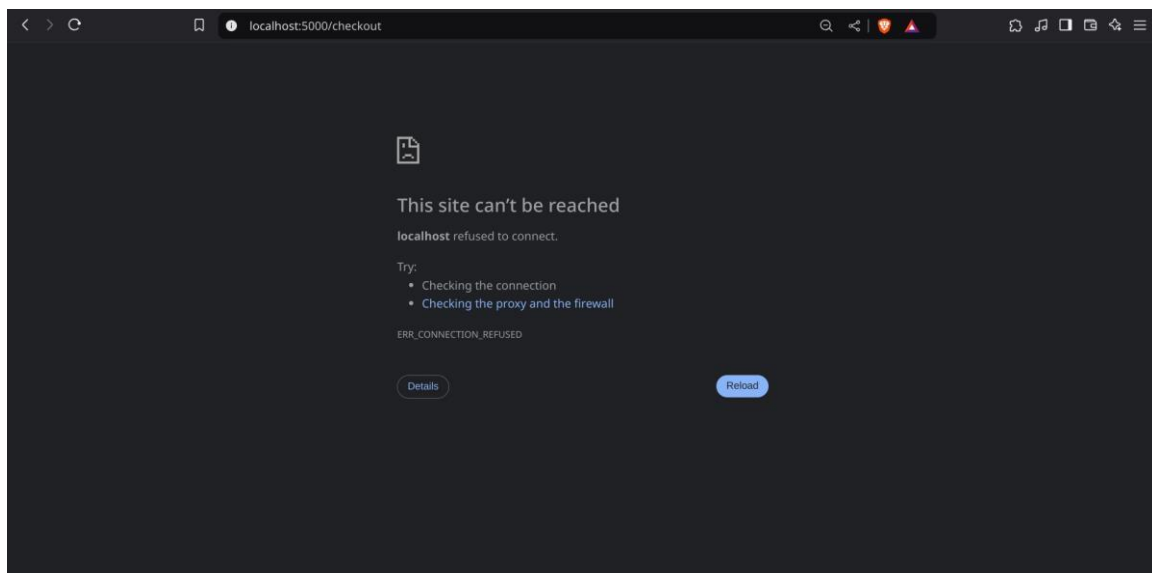
Name	Description	Quantity
Backpack	A durable and stylish backpack for daily use.	10
Wireless Mouse	A sleek and ergonomic wireless mouse with a long battery life.	20
Bluetooth Speaker	A portable Bluetooth speaker with high-quality sound and deep bass.	30
Laptop Stand	An adjustable laptop stand for better 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
Electric Kettle	A fast-boiling electric kettle with auto shut-off.	20
Smart Watch	A stylish smartwatch with fitness and notification features.	10
LED Desk Lamp	A modern LED desk lamp with adjustable brightness.	35
Portable Projector	A mini portable projector with HD resolution.	8

DESCRIPTION: Successful in fetching the products list after modifying the SRN in line no. 15 of main.py, registration and login.

d. When you are done adding to the cart, you can checkout.

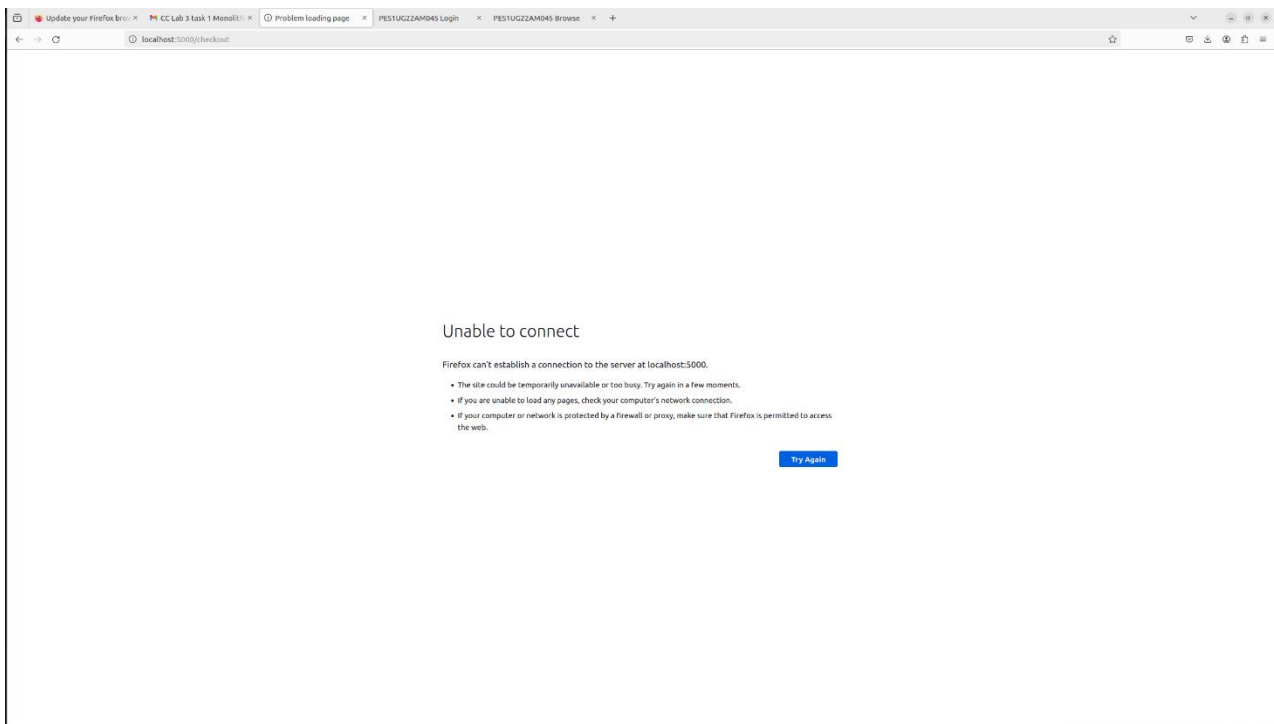
<http://localhost:5000/cart>

6. Time to checkout, you can click on checkout <http://localhost:5000/cart> here or you can visit <http://localhost:5000/checkout> . You will see this page , Take a Screenshot(SS2).



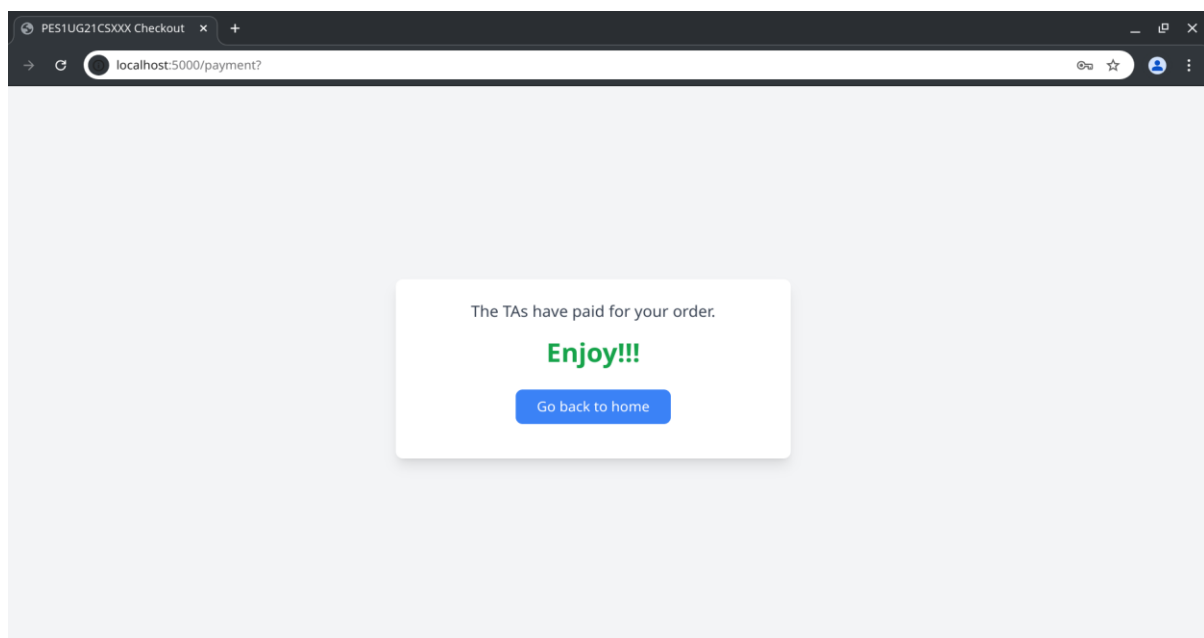
PTO

MY SCREENSHOT (SS2):

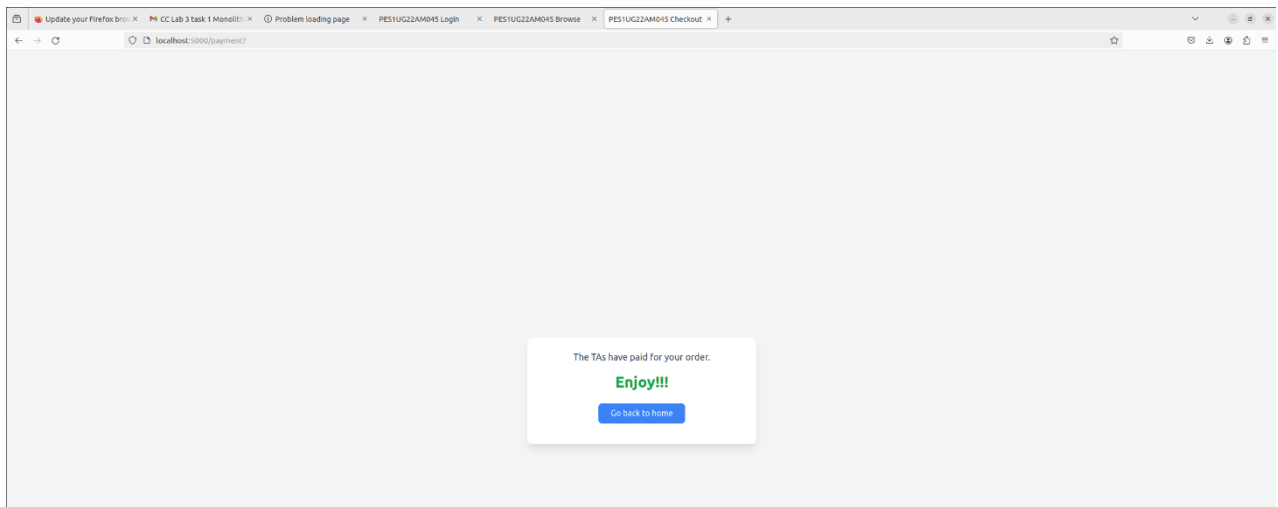


DESCRIPTION: The screenshot shows a "Unable to connect" error in Firefox while accessing <http://localhost:5000/checkout>, indicating that the local server on port 5000 is not running or blocked. Reasons include the bug present in the code.

- Well what happened, did the server crash, oops looks like we left a bug in there that brought the entire server down. This is one of the main disadvantage of the monolithic apps, even though other service can work because of one error whole thing came down, to avoid this we have microservices, which we will look into upcoming labs
- Let's fix the bug, go to `/checkout/__init__.py`, comment out line 16.
- Rerun the server
- And visit <http://localhost:5000/checkout>. This will be **Screenshot 3 (SS3)**. Make sure your **SRN** is clearly visible.



MY SCREENSHOT (SS3):



DESCRIPTION: Successful execution after commenting line no. 16 in checkout/__init__.py

Great, we have bug-free code (hopefully). You might think having no bugs is good, but that is not true, especially in monolithic apps where everything is merged together. We need to optimize the code.

How do we know if the code is doing any good or not? There are many tools for that, but we will look into one of them.

Locust - A load-testing tool, where you can define a test and run it on your endpoints.

Before you start you need to set the database, run *insert_product.py* (Make sure *main.py* is running parallelly for this to work)

```
python3 insert_product.py
```

Then you can run locust (Make sure to run locust in a terminal with the venv enabled and do not run the commands mentioned below inside the locust directory)

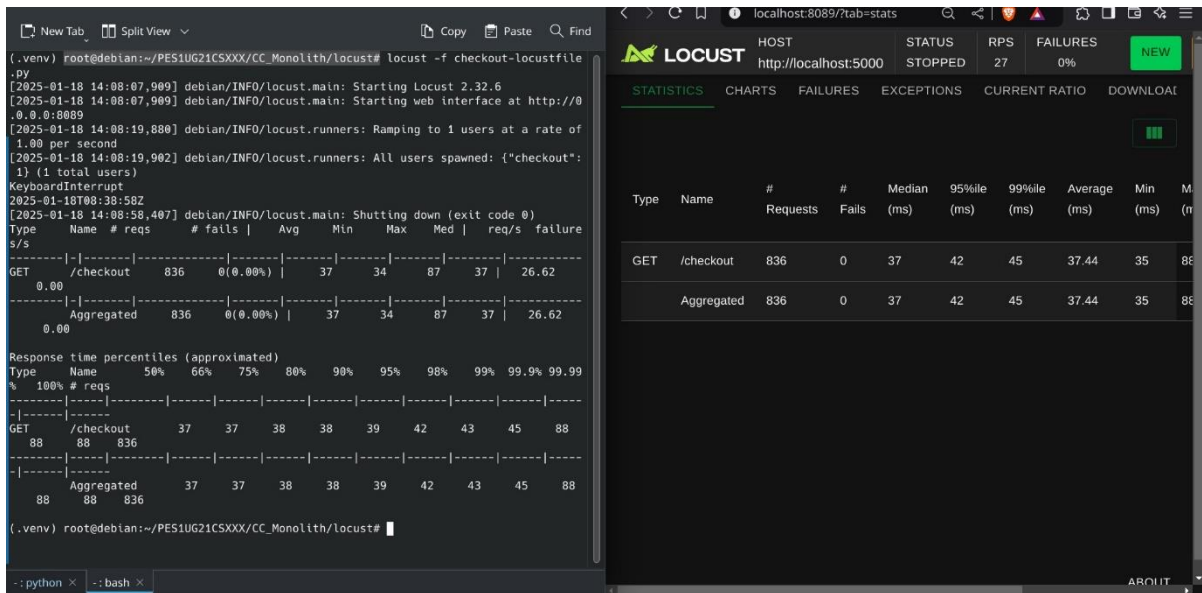
```
locust -f locust/checkout-locustfile.py
```

You can visit the site <http://localhost:8089/> (Make sure *main.py* is running)

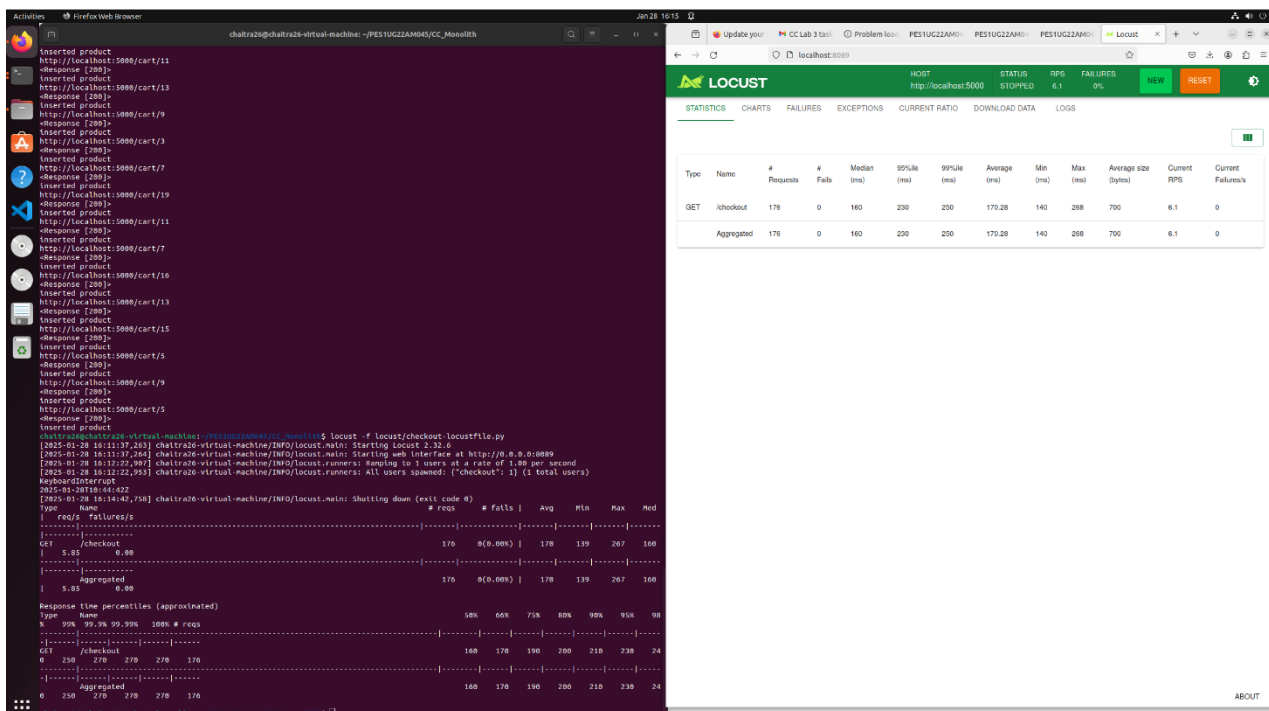
On the UI you can specify the **number of users** you want it to simulate, **Ramp up** is the number of users that will be started per second, under advanced you can set the time until the test runs.

1. Here we will try to optimise the /checkout route
 - a. The **locust directory** has the locust files which are already configured with the load tests.
 - b. Add user(1), Rampup(1) and measure the performance for 30 seconds and terminate locust running on your terminal. The **Screenshot (SS4)** expected here is a split screen of the terminal and the locust dashboard. Make sure your **SRN is visible in the terminal**. (If you have a very powerful machine, then you might not see any change in the numbers, so you can go ahead and increase the number of users, rampup and time)

2. Here we will try to optimise the /checkout route
 - a. The **locust directory** has the locust files which are already configured with the load tests.
 - b. Add user(1), Rampup(1) and measure the performance for 30 seconds and terminate locust running on your terminal. The **Screenshot (SS4)** expected here is a split screen of the terminal and the locust dashboard. Make sure your **SRN is visible in the terminal**. (If you have a very powerful machine, then you might not see any change in the numbers, so you can go ahead and increase the number of users, rampup and time)



MY SCREENSHOT (SS4):



DESCRIPTION: In this scenario, we are testing the server load using the Locust tool by configuring the parameters based on the specified metrics. It is important to note that the number of requests recorded here is 176. The next step involves optimizing the checkout code for better performance.

- c. We could probably optimise this.

3. Optimise the route `/checkout`

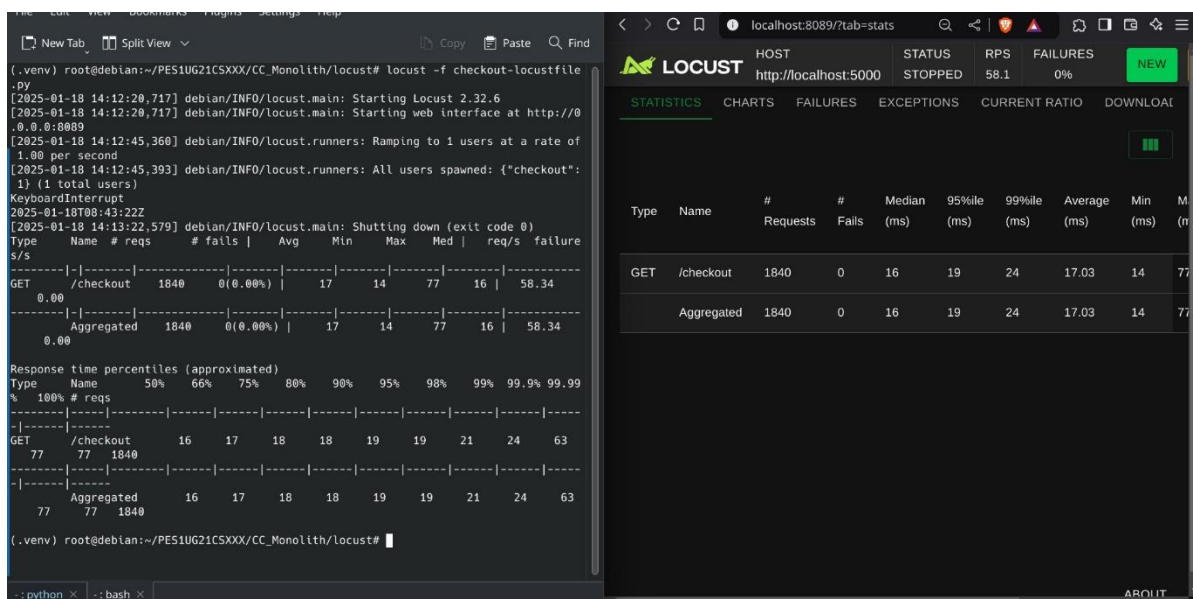
- a. So if we look at the `/checkout` handler on the `main.py` file at line no. 127, we see that the handler is fairly optimised, so there is not a lot we can do here.
- b. Next we will go through user defined functions, and under this only `chk` function exists which is an alias of `checkout` function imported from `checkout` module.
- c. Now if we go to `/checkout/__init__.py`, and look at the code we see that the code is bad.

```
for item in cart:
    while(item.cost > 0):
        total += 1
        item.cost -= 1
```

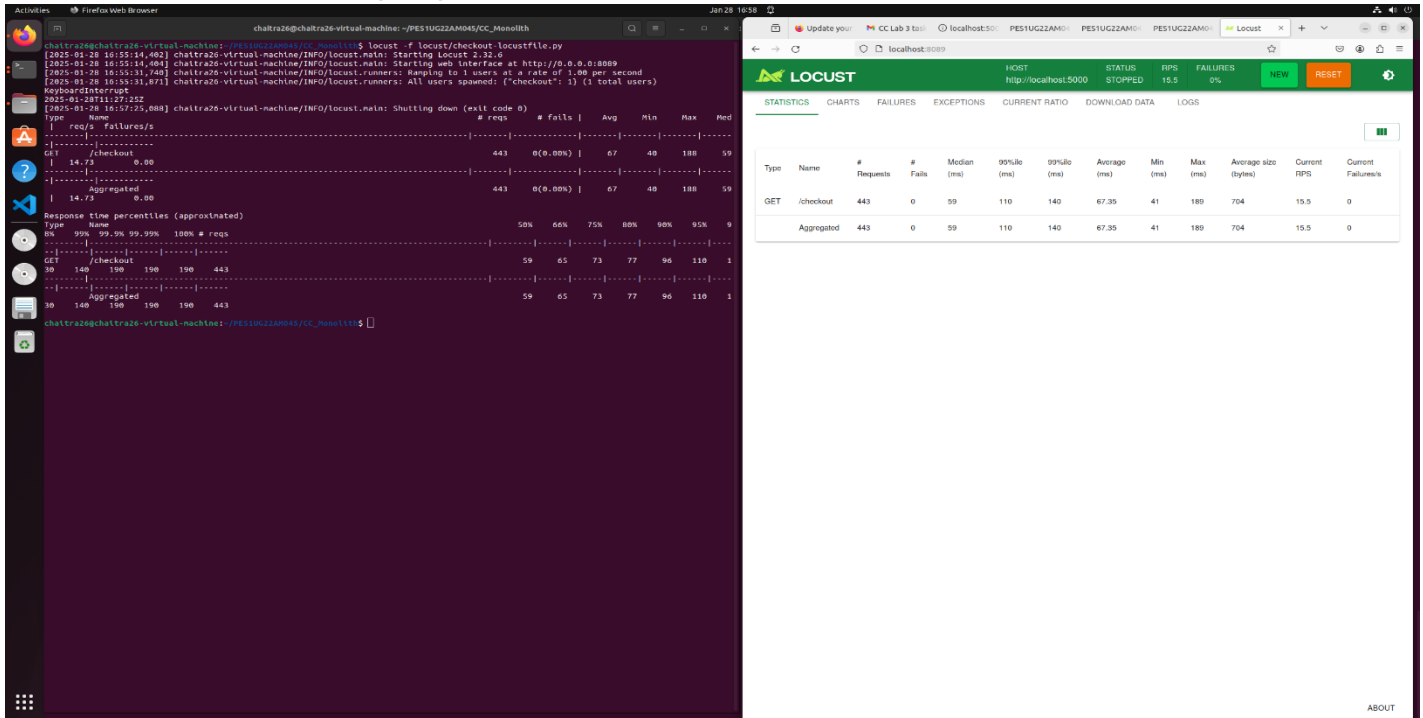
- d. The code can be replaced with

```
for item in cart:
    total += item.cost
```

- e. Now rerun locust and notice how it has significantly improved. Notice how the number of requests processed has doubled and the average time to handle them has decreased. The **Screenshot (SS5)** expected here is a split screen of the terminal and the locust dashboard. Make sure your **SRN is visible in the terminal**.



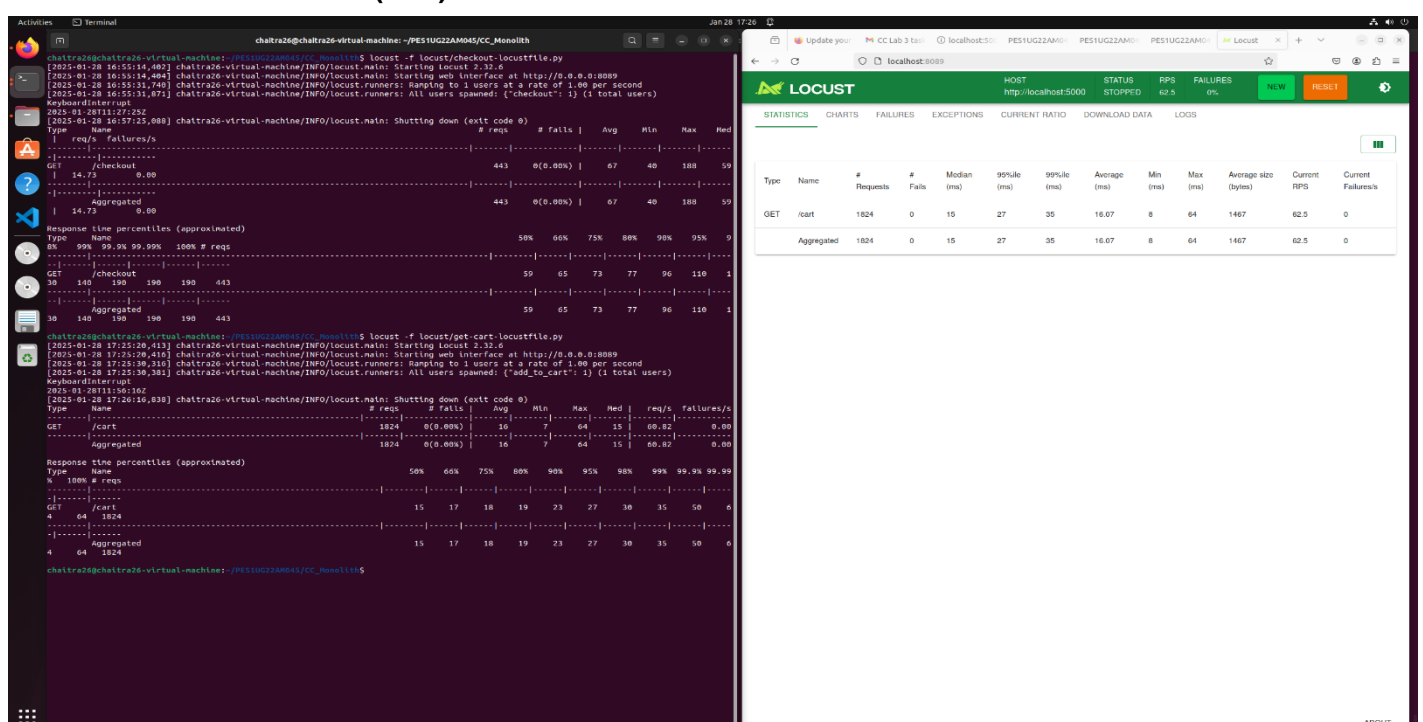
MY SCREENSHOT (SS5):



DESCRIPTION: After optimizing the checkout code, the number of requests has doubled compared to the previous version. This demonstrates a significant improvement in performance.

- Until this point, we have guided you through this lab, now we would like you to optimize the next 2 routes **/cart** and **/browse**, we have provided you with 2 more locust files(`get-cart-locustfile.py` and `browse-locustfile.py`) preconfigured with the test. Optimize the code. Similar to the optimization done in `/checkout` route, we expect you to take the **Screenshots (SS6 and SS7)** of **before and after** optimizing `/cart` . Similarly **Screenshots (SS8 and SS9)** of **before and after** optimizing `/browse`. Also we expect you to provide why and how you have optimised the code.

MY SCREENSHOT (SS6):



DESCRIPTION: This indicates the measure of cart requests before optimization. Observe that the number of requests here is 1824.

Old Code:

```
1  import json
2  import products
3  from cart import dao
4  from products import Product
5  class Cart:
6      def __init__(self, id: int, username: str, contents: list[Product], cost: float):
7          self.id = id
8          self.username = username
9          self.contents = contents
10         self.cost = cost
11
12         def load(data):
13             return Cart(data['id'], data['username'], data['contents'], data['cost'])
14     def get_cart(username: str) -> list:
15         cart_details = dao.get_cart(username)
16         if cart_details is None:
17             return []
18
19         items = []
20         for cart_detail in cart_details:
21             contents = cart_detail['contents']
22             evaluated_contents = eval(contents)
23             for content in evaluated_contents:
24                 items.append(content)
25
26         i2 = []
27         for i in items:
28             temp_product = products.get_product(i)
29             i2.append(temp_product)
30         return i2
31
32     def add_to_cart(username: str, product_id: int):
33         dao.add_to_cart(username, product_id)
34
35
36     def remove_from_cart(username: str, product_id: int):
37         dao.remove_from_cart(username, product_id)
38
39     def delete_cart(username: str):
40         dao.delete_cart(username)
```

New Code (Optimized to avoid unnecessary loops and redundant operations):

```
1  import json
2  from cart import dao
3  from products import Product, get_product
4  class Cart:
5      def __init__(self, id: int, username: str, contents: list[Product], cost: float):
6          self.id = id
7          self.username = username
8          self.contents = contents
9          self.cost = cost
10         @staticmethod
11         def load(data):
12             return Cart(data['id'], data['username'], data['contents'], data['cost'])
13     def get_cart(username: str) -> list[Product]:
14         """
15         Retrieves the user's cart and converts product IDs into Product objects.
16         """
17         cart_details = dao.get_cart(username)
18         if not cart_details:
19             return []
```

```

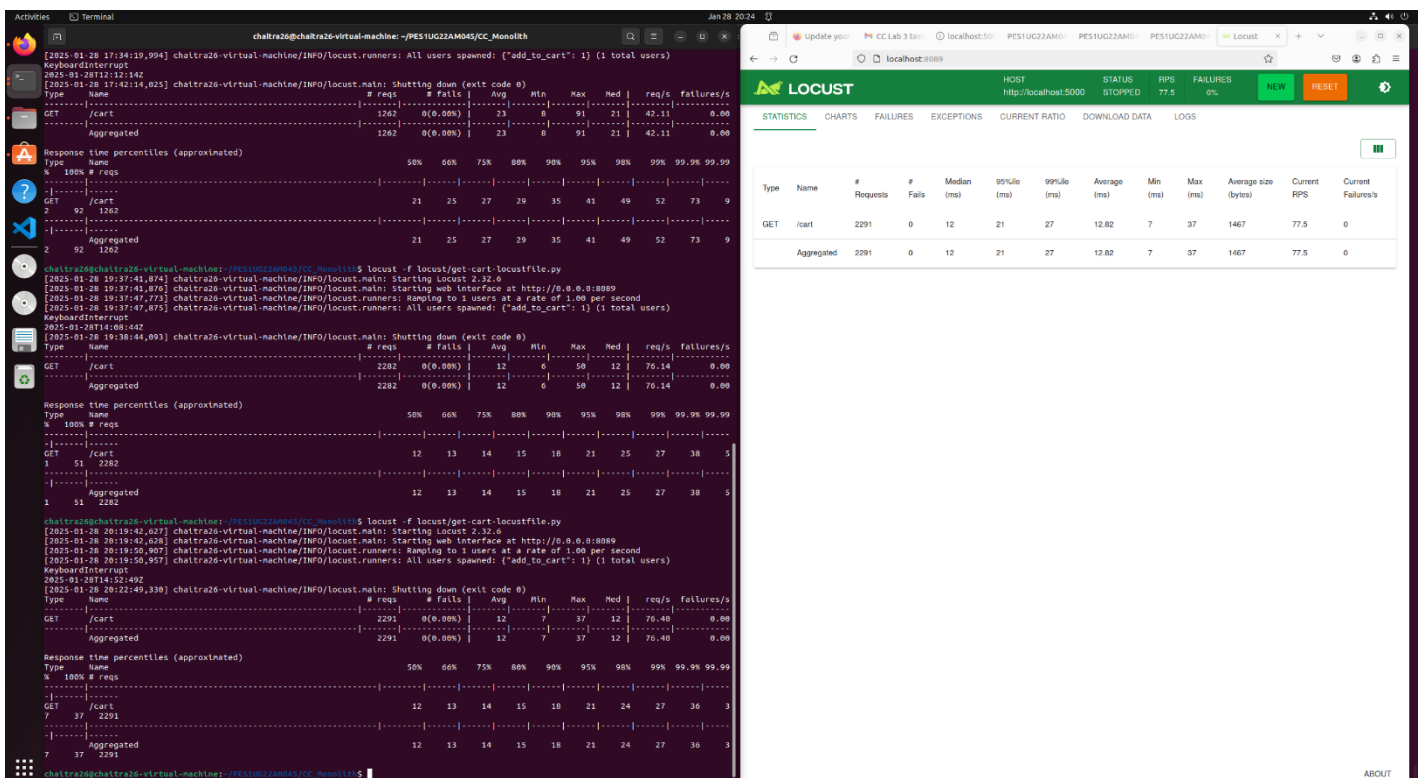
20     # Optimized to avoid unnecessary loops and redundant operations
21     items = [item for cart_detail in cart_details for item in json.loads(cart_detail['contents'])]
22     product_map = {item_id: get_product(item_id) for item_id in set(items)}
23     return [product_map[item_id] for item_id in items]
24 def add_to_cart(username: str, product_id: int):
25     """
26     Adds a product to the user's cart.
27     """
28     dao.add_to_cart(username, product_id)
29 def remove_from_cart(username: str, product_id: int):
30     """
31     Removes a specific product from the user's cart.
32     """
33     dao.remove_from_cart(username, product_id)
34 def delete_cart(username: str):
35     """
36     Deletes the user's entire cart.
37     """
38     dao.delete_cart(username)

```

Why and how you has the code been optimized:

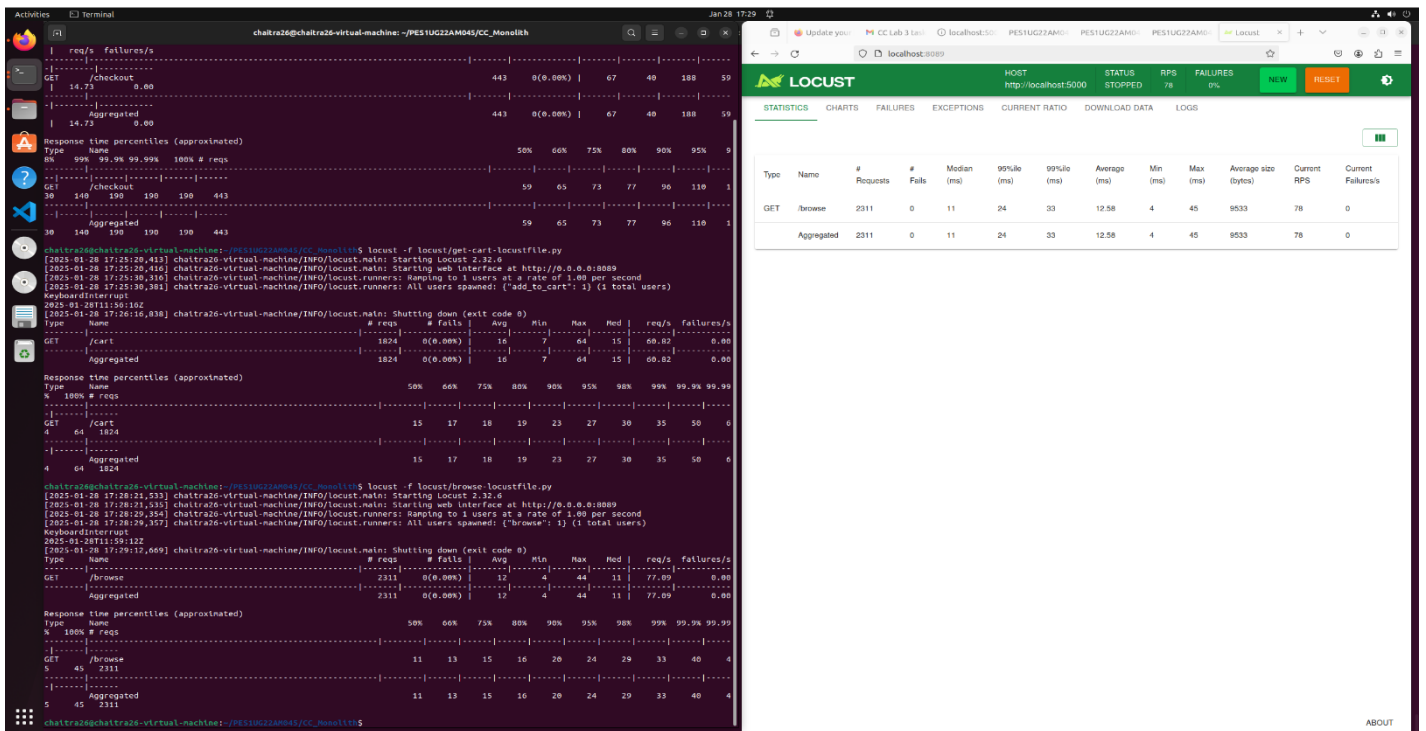
The optimized code improves performance, security, and readability. The load method is marked as @staticmethod for clarity, and eval is replaced with json.loads to safely parse contents. Nested loops are combined into a single list comprehension for efficiency, and a product_map dictionary is used to cache product lookups, reducing redundant calls to get_product. Type hints and docstrings enhance clarity, and unused imports are removed. These changes result in faster, safer, and more maintainable code.

MY SCREENSHOT (SS7):



DESCRIPTION: This indicates the performance after optimizing the cart code. Observe that the number of requests has increased compared to the previous measurement.

MY SCREENSHOT (SS8):



DESCRIPTION: This is the measure of the browse (products) code before optimization. Notice that the number of requests here is 2311.

Old Code:

```

1  from products import dao
2  class Product:
3      def __init__(self, id: int, name: str, description: str, cost: float, qty: int = 0):
4          self.id = id
5          self.name = name
6          self.description = description
7          self.cost = cost
8          self.qty = qty
9      def load(data):
10         return Product(data['id'], data['name'], data['description'], data['cost'], data['qty'])
11 def list_products() -> list[Product]:
12     products = dao.list_products()
13     result = []
14     for product in products:
15         result.append(Product.load(product))
16     return result
17 def get_product(product_id: int) -> Product:
18     return Product.load(dao.get_product(product_id))
19 def add_product(product: dict):
20     dao.add_product(product)
21 def update_qty(product_id: int, qty: int):
22     if qty < 0:
23         raise ValueError('Quantity cannot be negative')
24     dao.update_qty(product_id, qty)

```

New Code (Optimized to avoid unnecessary loops and redundant operations):

```

1  from products import dao
2  class Product:
3      def __init__(self, id: int, name: str, description: str, cost: float, qty: int = 0):
4          self.id = id
5          self.name = name
6          self.description = description
7          self.cost = cost
8          self.qty = qty
9      @staticmethod
10     def load(data):
11         return Product(data['id'], data['name'], data['description'], data['cost'], data['qty'])
12 def list_products() -> list[Product]:
13     """
14     Retrieves and returns a list of Product objects efficiently.
15     """
16     products = dao.list_products()
17     return [Product.load(product) for product in products]
18 def get_product(product_id: int) -> Product:
19     """
20     Retrieves a single product by its ID and converts it to a Product object.
21     """
22     return Product.load(dao.get_product(product_id))
23 def add_product(product: dict):
24     """
25     Adds a new product using the provided product dictionary.
26     """
27     dao.add_product(product)
28 def update_qty(product_id: int, qty: int):
29     """
30     Updates the quantity of a product, ensuring the quantity is non-negative.
31     """
32     if qty < 0:
33         raise ValueError('Quantity cannot be negative')
34     dao.update_qty(product_id, qty)

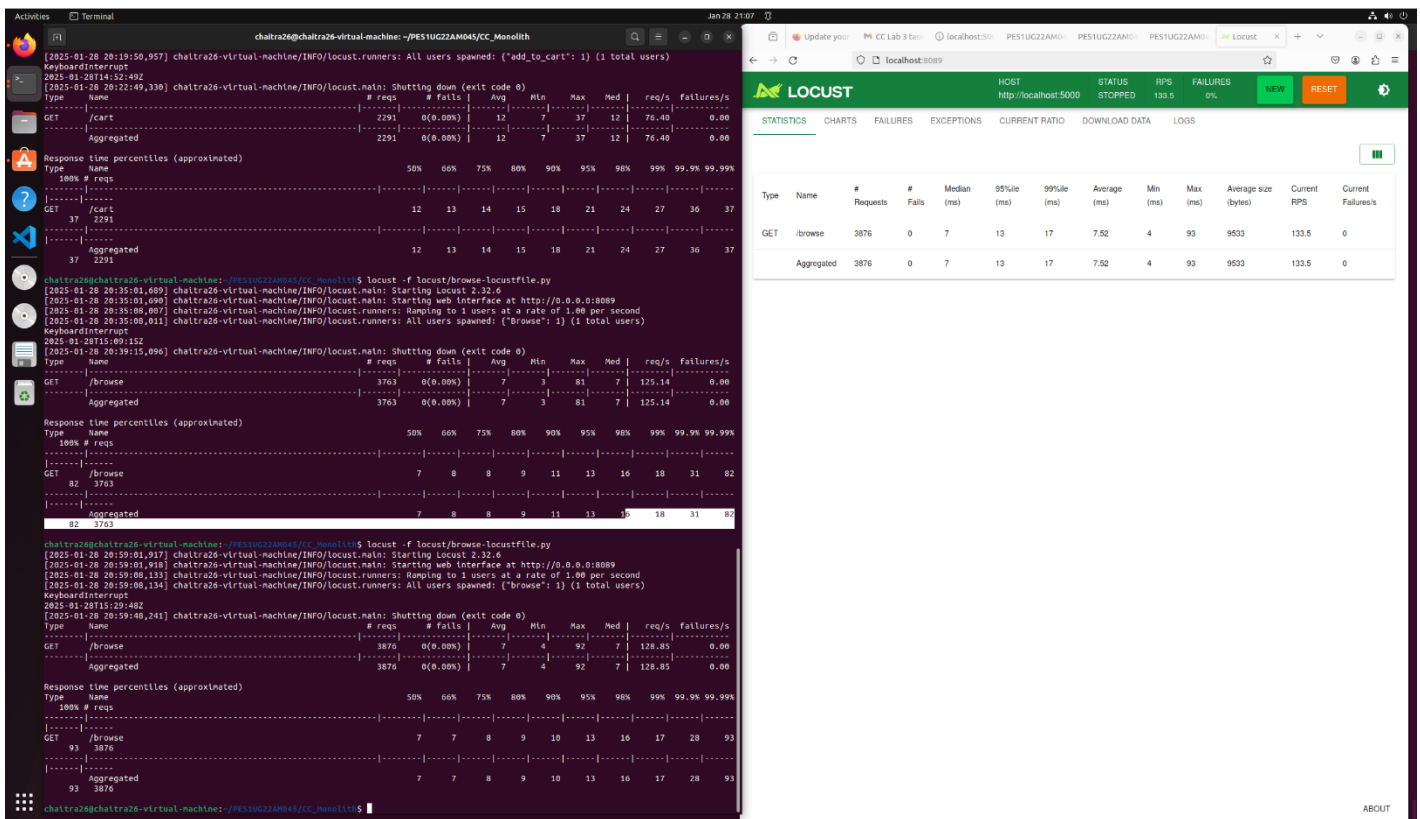
```

Why and how you has the code been optimized:

The optimized code introduces several improvements in terms of clarity, efficiency, and maintainability. Firstly, the load method has been decorated with @staticmethod since it does not rely on instance data, making it clearer that it's a utility method. The list_products function has been optimized by using a list comprehension, which is more concise and efficient compared to manually appending items in a loop. The get_product and list_products functions now include docstrings, which improve readability and help clarify their purpose. Overall, these changes make the code more efficient, maintainable, and easier to understand.

PTO

MY SCREENSHOT (SS9):



DESCRIPTION: These are the measures of the browse (products) code after optimization. Notice the significant increase in the number of requests.

- Finally, make a GitHub repository and upload the modified code there, make sure the repository is public, and submit the link while submitting the lab.

GitHub repository link - <https://github.com/chaitra-v26/Monolithic-Architecture-Cloud-Computing-Lab-03->