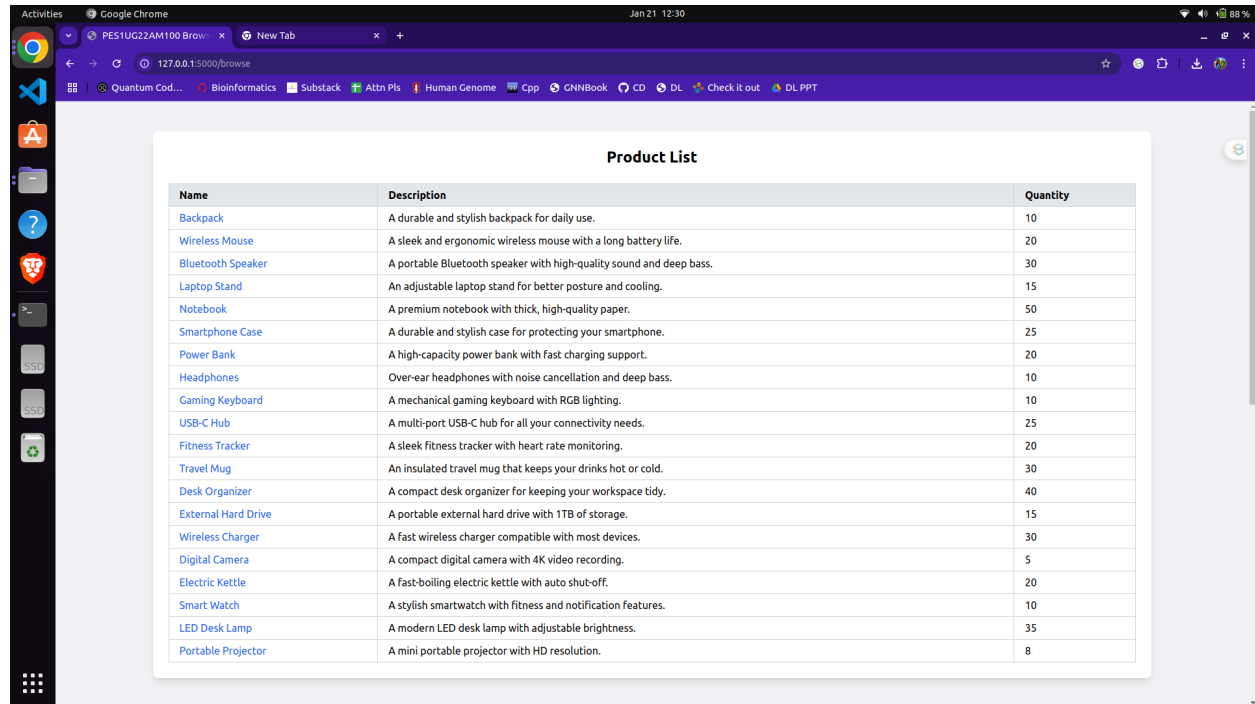


CC Lab 3- Jan 16th, 2025

1. Visiting the Website

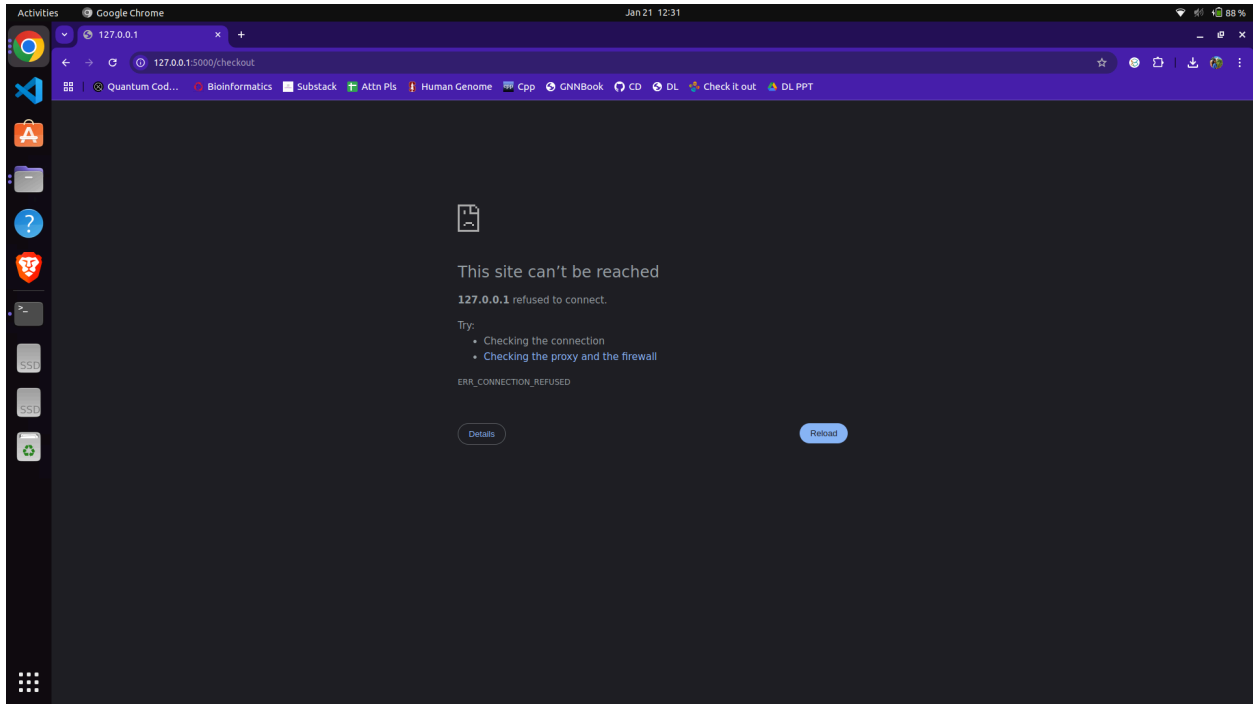


The screenshot shows a web browser window with a single tab titled 'PES1UG22AM100 Brow...'. The address bar displays '127.0.0.1:5000/browse'. The browser's bookmark bar contains several items: Quantum Cod..., Bioinformatics, Substack, Attn Pls, Human Genome, Cpp, GNNBook, CD, DL, Check it out, and DL PPT. The main content area displays a 'Product List' table with three columns: Name, Description, and Quantity. The table lists 20 products, each with a unique name, a brief description, and a quantity value. The browser's status bar at the bottom shows the date and time as 'Jan 21 12:30' and a battery level of 88%.

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

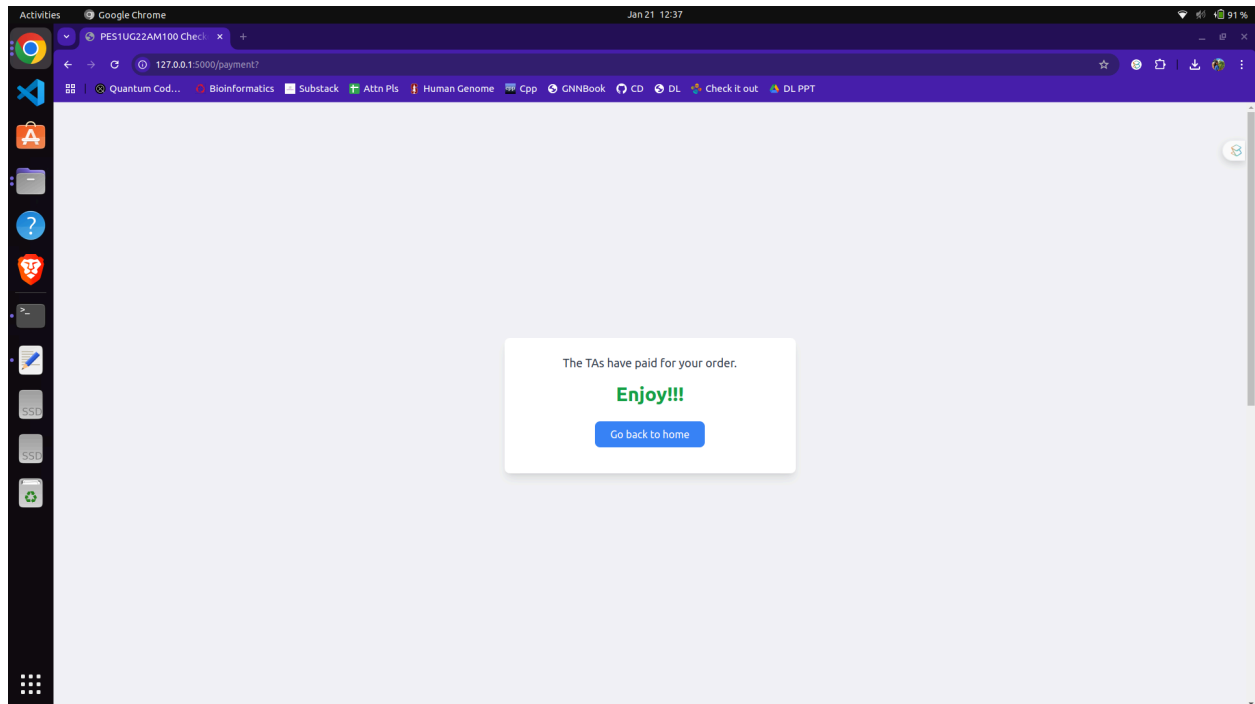
This screenshot shows the product browsing page (/browse) of the monolithic application after successfully adding a product to the cart. It highlights the functionality where users can view product details and add items to their cart. My SRN (AM100) is visible in the interface to indicate the setup is personalized and running correctly.

2. Going to Checkout and Site Crash



This screenshot captures the checkout page (/checkout) before fixing the bug in the monolithic application. The error caused the server to crash, demonstrating the drawback of monolithic systems where a single failure can disrupt the entire application. Now this error was simulated, but still represents a real-life scenario where monolithic applications can seriously fail, and thereby shows us a big drawback of monolithic applications/

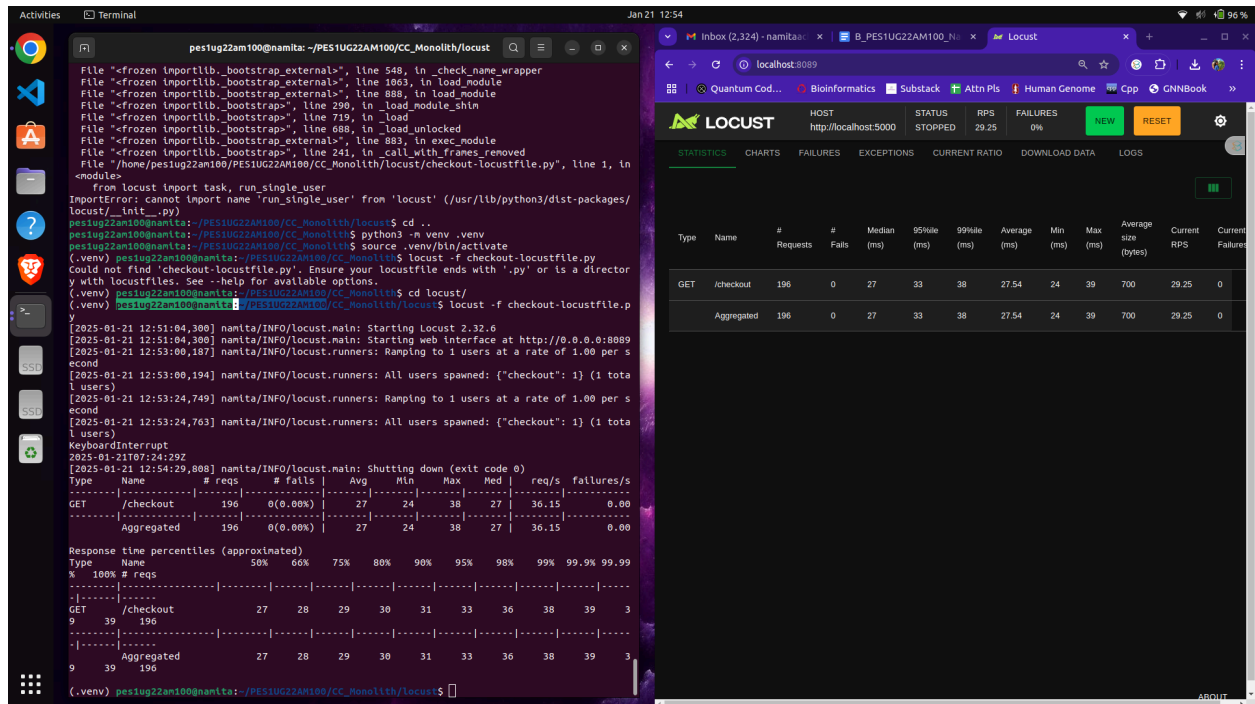
3. Commenting Out Line 15



This screenshot shows the corrected checkout page (/checkout) after fixing the bug in the code. It illustrates a successful transaction flow in the monolithic application. What commenting the bug out did, is basically allowed the application to continue executing without abruptly terminating when an error occurred.

By removing the `os._exit(1)` line, we enabled the program to handle exceptions more gracefully. Instead of crashing, the application can now provide meaningful feedback to the user.

4. Running Locust on Checkout Before Optimizing



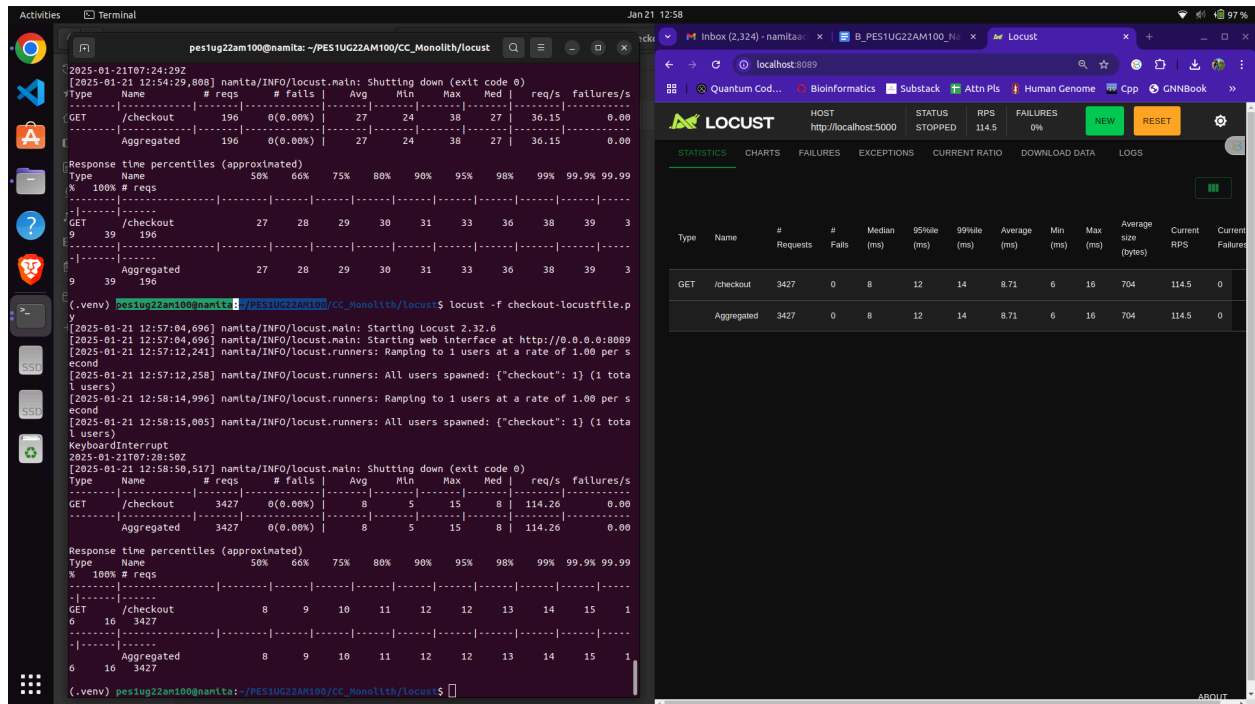
Locust is a tool for load testing. We're using it to simulate multiple users interacting with a web application at the same time. The goal is to measure how the application performs under stress—things like how long it takes to respond to requests and how it handles high traffic. Essentially, we're trying to find bottlenecks or weaknesses before they cause real problems. That's what we're essentially doing across screenshots

Where the unoptimized code could have improved:

- Used an inefficient while loop to calculate total, modifying item.cost
- Abruptly terminated the program with `os._exit(1)`

Requests: 196

5. Running Locust on Checkout After Optimizing

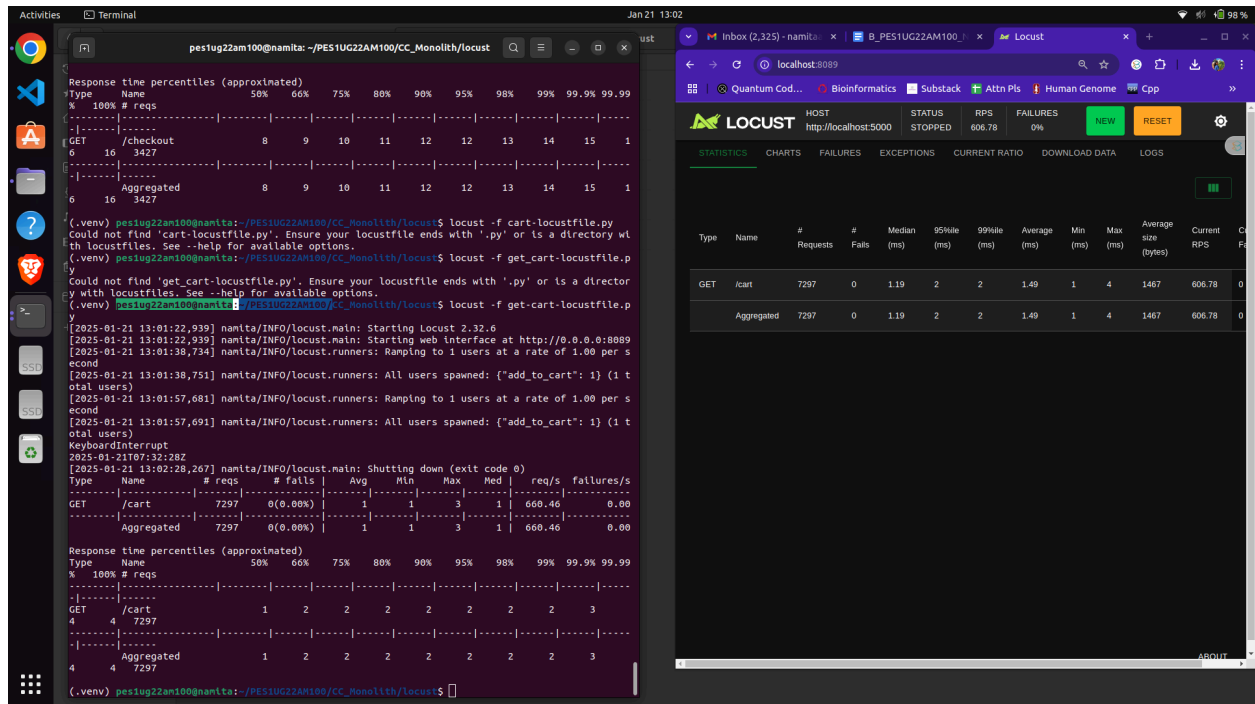


Where it was improved in the optimized code and why:

- Replaced the loop with total += item.cost, preserving item.cost and improving efficiency.
- Commented out os._exit(1), allowing proper program termination.

Requests: 3427

6. Running Locust on Cart Before Optimizing

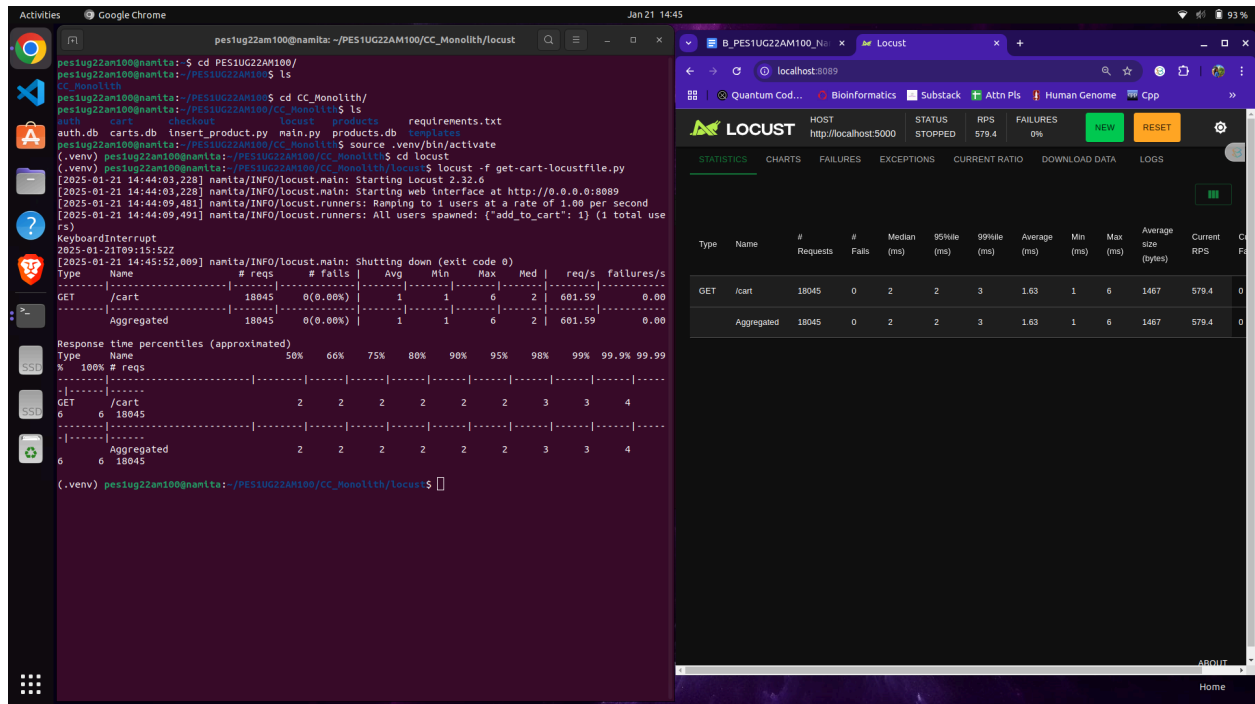


Where the unoptimized code could have improved:

- Using eval to process contents: The unoptimized code uses eval, which is risky and inefficient for parsing cart contents.
- Redundant looping and inefficient error handling: It loops twice over cart_details and lacks robust error handling for issues like malformed data.

Requests: 7297

7. Running Locust on Cart After Optimizing

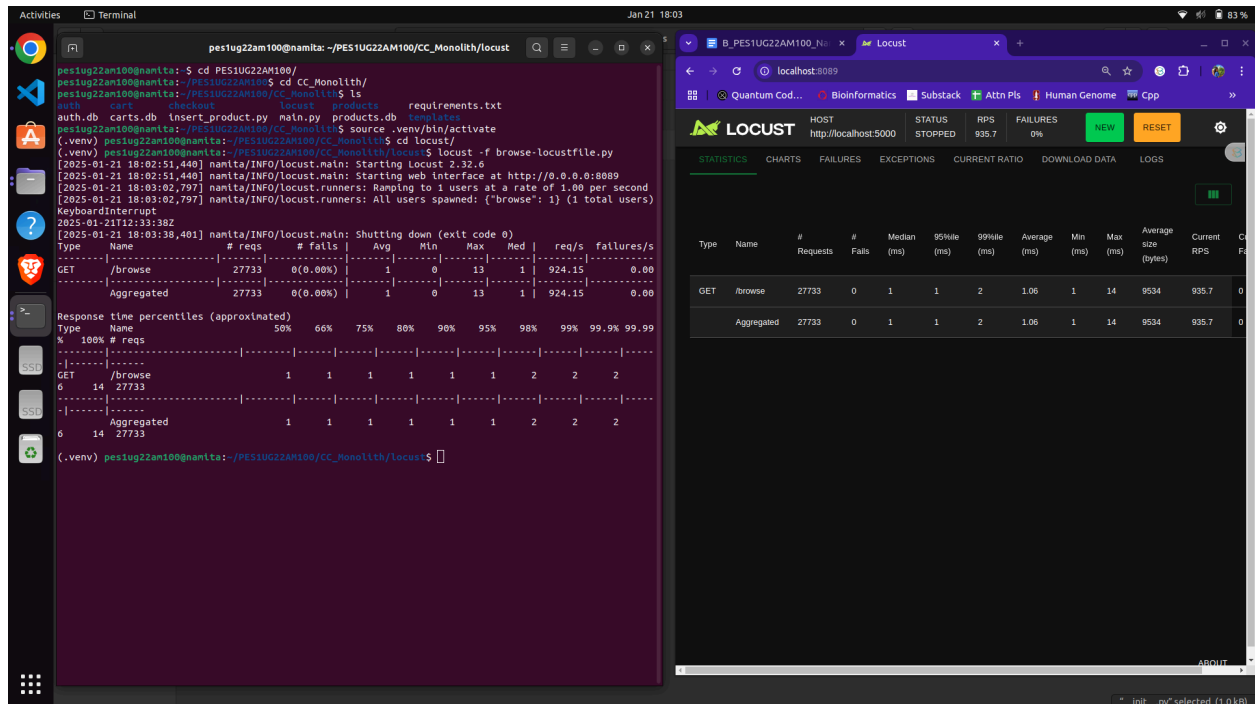


Where it was improved in the optimized code and why:

- Replaced eval with json.loads: The optimized code uses json.loads, which is safer and more efficient for parsing the cart contents.
- Streamlined processing and added error handling: The optimized code processes cart contents in a single loop and adds proper error handling for potential issues.

Requests: 18045

6. Running Locust on Browse Before Optimizing

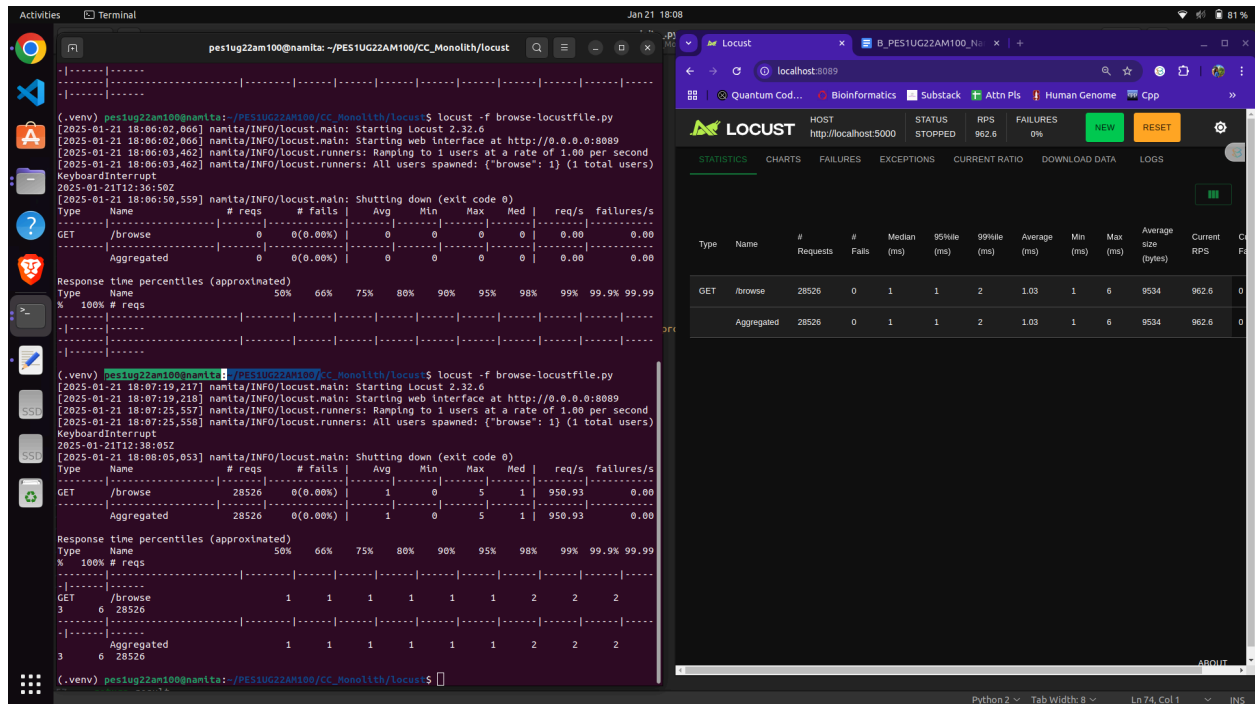


Where the unoptimized code could have improved:

- load method not marked as static: The unoptimized code uses a regular method for load, which is not ideal for a utility function that doesn't need instance access.
- Manual list population in list_products: It manually loops to append products, which is less concise and Pythonic than list comprehensions.
- No validation in add_product: The unoptimized code doesn't validate the input product dictionary, potentially leading to missing required fields.

Requests: 27733

7. Running Locust on Browse After Optimizing



Where it was improved in the optimized code and why:

- Static load method: The optimized code marks load as `@staticmethod`, making it more appropriate for a utility function that doesn't rely on instance attributes.
- List comprehension in `list_products`: The optimized code uses a list comprehension to directly create the list of products, making it more concise and readable.
- Validation in `add_product`: The optimized code adds validation to ensure that the required fields (id, name, description, cost) are present in the product dictionary, preventing potential errors.

Requests: 28526