Lab3
Outputs

Simple Test Cases

First, we tested if the **catalog service** works as expected by sending the following lookup(). The test cases in test_catalog.py include:
- 5 lookup requests with valid toy names
- 4 lookup requests with invalid names
- 1 lookup request with invalid URL format.
- Restock by 100 when quantity is less than 10, every 10 secs

```
URL:  http://localhost:4551/product/Barbie
{"data": {"name": "Barbie", "price": 89.0, "quantity": 1950}}
URL:  http://localhost:4551/product/Fox
{"data": {"name": "Fox", "price": 20.99, "quantity": 1274}}
URL:  http://localhost:4551/product/Tux
{"data": {"name": "Tux", "price": 25.99, "quantity": 2140}}
URL:  http://localhost:4551/product/Monopoly
{"data": {"name": "Monopoly", "price": 101.2, "quantity": 2200}}
URL:  http://localhost:4551/product/Elephant
{"data": {"name": "Elephant", "price": 25.99, "quantity": 2200}}
URL:  http://localhost:4551/product/Book
Sorry Book not found

URL:  http://localhost:4551/product/Pen
Sorry Pen not found

URL:  http://localhost:4551/product/Teddy
Sorry Teddy not found

URL:  http://localhost:4551/product/Train
Sorry Train not found

URL:  http://localhost:4551/Teddy/Lego
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/urllib3/connectionpool.py", line 793, in urlopen
    response = self._make_request(
               ^^^^^^^^^^^^^^^^^^
```

Initial stock

catalog.csv

src > Catalog > catalog.csv

```
1   Toy,Quantity,Price
2   Tux,100,25.99
3   Whale,100,10.89
4   Dolphin,100,15.99
5   Monopoly,100,101.2
6   Fox,100,20.99
7   Elephant,100,25.99
8   Python,100,22.1
9   Barbie,100,89.0
10  Lego,100,22.0
11  Marbles,100,36.78
```

In between

c > Catalog > catalog.csv

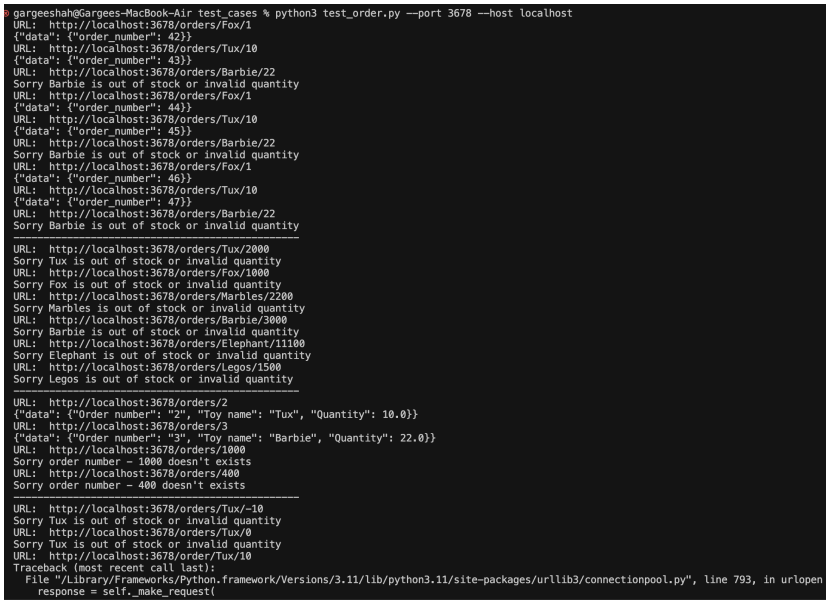
```
1   Toy,Quantity,Price
2   Tux,28,25.99
3   Whale,74,10.89
4   Dolphin,6,15.99
5   Monopoly,13,101.2
6   Fox,31,20.99
7   Elephant,12,25.99
8   Python,100,22.1
9   Barbie,96,89.0
10  Lego,12,22.0
11  Marbles,61,36.78
```

After 10 sec

c > Catalog > catalog.csv

```
1   Toy,Quantity,Price
2   Tux,28,25.99
3   Whale,60,10.89
4   Dolphin,106,15.99
5   Monopoly,13,101.2
6   Fox,31,20.99
7   Elephant,12,25.99
8   Python,92,22.1
9   Barbie,82,89.0
0   Lego,12,22.0
1   Marbles,61,36.78
```

Secondly, we tested **both the catalog and order services**. The test cases in test_order.py include:
- 3 legal order requests. Do this 3 times
- 2 legal order lookup requests and 2 illegal lookup requests (order number not found)
- 1 invalid URL order request
- 7 excessive quantity messages.



Finally, we tested the **whole application**. The test cases include:

- 5 legal lookup requests
- 3 lookup requests with non-existent toy names
- 3 order operations with valid quantity
- 6 order operations with excessive quantity
- 2 order operations with invalid quantity
- 2 legal order lookup requests and 2 illegal lookup requests (order number not found)

```
URL:  http://localhost:4773/product/Barbie
{"data": {"name": "Barbie", "price": 89.0, "quantity": 1472}}
URL:  http://localhost:4773/product/Fox
{"data": {"name": "Fox", "price": 20.99, "quantity": 775}}
URL:  http://localhost:4773/product/Tux
{"data": {"name": "Tux", "price": 25.99, "quantity": 1650}}
URL:  http://localhost:4773/product/Monopoly
{"data": {"name": "Monopoly", "price": 101.2, "quantity": 1700}}
URL:  http://localhost:4773/product/Elephant
{"data": {"name": "Elephant", "price": 25.99, "quantity": 1700}}
URL:  http://localhost:4773/product/Book
{"error": {"code": 404, "message": "Toy not found"}}
URL:  http://localhost:4773/product/Pen
{"error": {"code": 404, "message": "Toy not found"}}
URL:  http://localhost:4773/product/Teddy
{"error": {"code": 404, "message": "Toy not found"}}
URL:  http://localhost:4773/product/Train
{"error": {"code": 404, "message": "Toy not found"}}
------------------------------------------------
URL:  http://localhost:4773/orders/Fox/1
http://localhost:4773/orders/Fox/1
{"data": {"order_number": 29}}
{"data": {"order_number": 29}}
URL:  http://localhost:4773/orders/Tux/10
http://localhost:4773/orders/Tux/10
{"data": {"order_number": 30}}
{"data": {"order_number": 30}}
URL:  http://localhost:4773/orders/Barbie/22
http://localhost:4773/orders/Barbie/22
{"data": {"order_number": 31}}
{"data": {"order_number": 31}}
------------------------------------------------
URL:  http://localhost:4773/orders/2
{"data": {"Order number": "2", "Toy name": "Tux", "Quantity": 10.0}}
URL:  http://localhost:4773/orders/3
{"data": {"Order number": "3", "Toy name": "Barbie", "Quantity": 22.0}}
URL:  http://localhost:4773/orders/1000
{"error": {"code": 400, "message": "Order number doesn't exist"}}
URL:  http://localhost:4773/orders/400
{"error": {"code": 400, "message": "Order number doesn't exist"}}
------------------------------------------------
URL:  http://localhost:4773/orders/Tux/2000
http://localhost:4773/orders/Tux/2000
Sorry Tux is out of stock or invalid quantity
{"error": {"code": 400, "message": "Sorry Tux is out of stock or invalid quantity"}}
URL:  http://localhost:4773/orders/Fox/1000
http://localhost:4773/orders/Fox/1000
Sorry Fox is out of stock or invalid quantity
{"error": {"code": 400, "message": "Sorry Fox is out of stock or invalid quantity"}}
URL:  http://localhost:4773/orders/Marbles/2200
http://localhost:4773/orders/Marbles/2200
{"data": {"order_number": 32}}
```

Testing by deploying the application on AWS

The following screenshots demonstrate the process of testing our application on the AWS t2.micro instance

To start out testing, we opened terminals on the instance, one for the catalog frontend, order and another one to crash the order process in between. The client is running on my local machine.

```
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ kill 8221
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ lsof -i :4551
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 8216 ubuntu    3u  IPv4  36795      0t0  TCP *:4551 (LISTEN)
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ kill 8216
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ cd ..
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401$ git pull origin main
Username for 'https://github.com': Gargeeshah
Password for 'https://Gargeeshah@github.com':
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 7 (delta 4), reused 7 (delta 4), pack-reused 0
Unpacking objects: 100% (7/7), done.
From https://github.com/umass-cs677-current/spring24-lab3-Gargeeshah-vaishnavi0401
 * branch            main       -> FETCH_HEAD
   b12d147..d0f3c06  main       -> origin/main
Updating b12d147..d0f3c06
Fast-forward
 src/Catalog/catalog.csv | 2 ++
 src/Client/client.py    | 4 ++--
 2 files changed, 4 insertions(+), 2 deletions(-)
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401$ cd src
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ ./run-all.sh
Running catalog.py
Running order replicas
Python script started on Port 1678 and Host localhost initialised with Order ID 3
Python script started on Port 2678 and Host localhost initialised with Order ID 2
Python script started on Port 3678 and Host localhost initialised with Order ID 1
CSV File name order2.csv
CSV File name order1.csv
Catalog Host: localhost Catalog Port 4551 order_PORT:  3678
Catalog Host: localhost Catalog Port 4551 order_PORT:  2678
follower_addresses ['http://localhost:3678', 'http://localhost:2678', 'http://localhost:1678']
order_number, current_server_address:  0 http://localhost:3678
follower_addresses ['http://localhost:3678', 'http://localhost:2678', 'http://localhost:1678']
order_number, current_server_address:  0 http://localhost:2678
Serving on port 3678
Serving on port 2678
CSV File name order3.csv
Catalog Host: localhost Catalog Port 4551 order_PORT:  1678
follower_addresses ['http://localhost:3678', 'http://localhost:2678', 'http://localhost:1678']
order_number, current_server_address:  0 http://localhost:1678
Serving on port 1678
CACHE:  True
Serving on port 4551
Running frontend-service.py
Cache request:  True
LEADER SELECTION
http://localhost:1678/ping
Health check and normal
127.0.0.1 - - [02/May/2024 05:34:01] "GET /ping?message=Ping HTTP/1.1" 200 -
{'message': 'Order service is responsive'}
You win
I am the leader now.
127.0.0.1 - - [02/May/2024 05:34:01] "POST /leaderselection HTTP/1.1" 200 -
Response from leader: {'message': 'From Leader: Leader ID 3 selected.'}
Order ID 3 is the leader now.
127.0.0.1 - - [02/May/2024 05:34:01] "POST /inform_replica HTTP/1.1" 200 -
Response from replicas {'message': 'From replica: Leader ID 3 is now leader.'}
Order ID 3 is the leader now.
127.0.0.1 - - [02/May/2024 05:34:01] "POST /inform_replica HTTP/1.1" 200 -
Response from replicas {'message': 'From replica: Leader ID 3 is now leader.'}
Serving on port 4773
```

Crash failure by killing the leader

Then, we ran multiple clients currently using $ ./run-all.sh on local. During sending the requests, we shut down the order server with ID=3 (the highest ID) -

```
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src/Order$ cd ..
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ lsof -i :1678
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 14838 ubuntu    3u  IPv4 257466      0t0  TCP *:1678 (LISTEN)
[ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ kill 14838
ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$
```

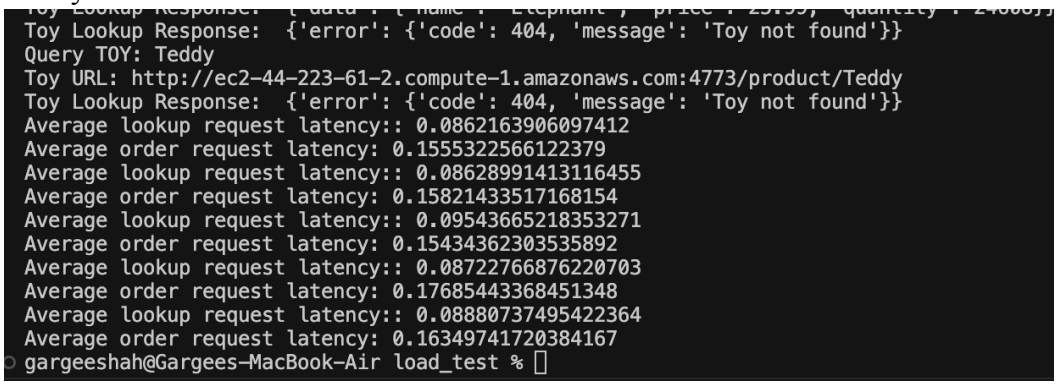
Order service 3 shut down

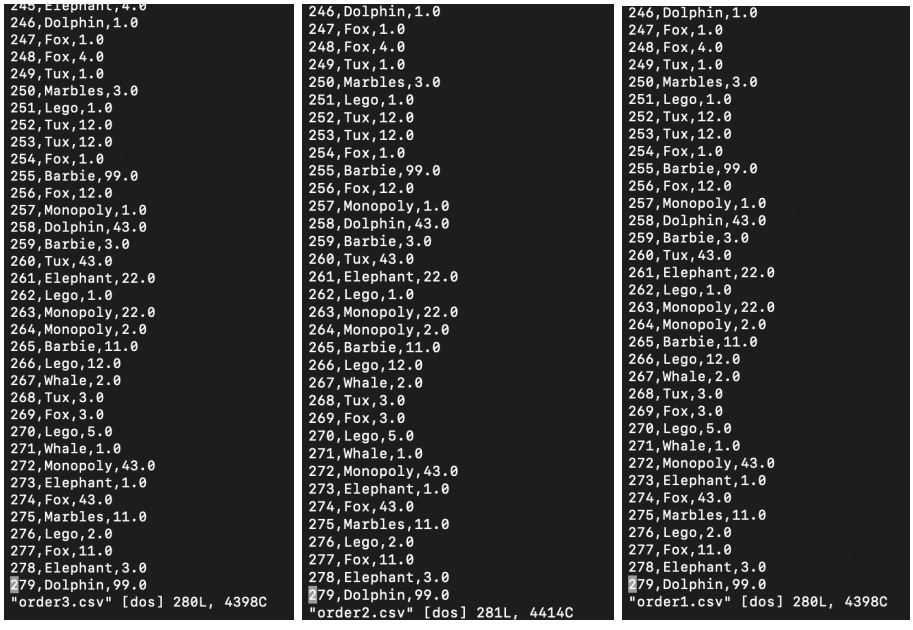
```
Order Number,Toy Name,Quantity
1,Whale,10.0
2,Monopoly,10.0
3,Whale,43.0
4,Dolphin,1.0
5,Lego,99.0
6,Elephant,11.0
7,Barbie,43.0
8,Marbles,2.0
~
~
~
~
```

Notice that now the node with ID=2 becomes the leader. The
ID=1 and ID=3 become the follower nodes which receive propagation requests from the leader

Then, we resumed the order ID=3 -

After the clients exit, check if there are any error messages showing on the client's terminal. If not, that means the order information retrieved from the order database are the same with the locally stored order information and also all the order databases are consistent.





Crash failure by killing one of the replica

Now, during sending the requests, we shut down the order server with ID= 1 -

```
ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ lsof -i :3678
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 8846 ubuntu    3u  IPv4 246704      0t0  TCP *:3678 (LISTEN)
ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src$ kill 8846
```

Order service 1 shut down

```
Error synchronizing order at http://localhost:3678: HTTPConnectionPool(host='localhost', port=3678): Max retries exceeded with url: /synchronize (Caused by NewConnectionError('<urllib3.connection.HTTPConnecti
on object at 0x7fcc3437a080>: Failed to establish a new connection: [Errno 111] Connection refused',))
127.0.0.1 - - [02/May/2024 07:48:31] "POST /orders/Fox/4.0 HTTP/1.1" 200 -
127.0.0.1 - - [02/May/2024 07:48:31] "POST /invalidate?toy=Fox HTTP/1.1" 200 -
127.0.0.1 - - [02/May/2024 07:48:31] "POST /invalidate?toy=Fox HTTP/1.1" 200 -
```

Then, we resumed the order ID=1 -

```
ubuntu@ip-172-31-31-124:~/spring24-lab3-Gargeeshah-vaishnavi0401/src/Order$ python3 order.py --port 3678 --host localhost --unique_id 1
CSV File name order1.csv
Catalog Host: localhost Catalog Port 4551 order_PORT:  3678
follower_addresses ['http://localhost:3678', 'http://localhost:2678', 'http://localhost:1678']
order_number, current_server_address:  315 http://localhost:3678
Serving on port 3678
Replica at http://localhost:3678 is not available. Skipping synchronization.
Not equal, checking sync
Sync-ing http://localhost:3678 315 435
Replica synchronized successfully
```

After the clients exit, check if there are any error messages showing on the client's terminal. If not, that means the order information retrieved from the order database are the same with the locally stored order information and also all the order databases are consistent.

```
540,Dolphin,2.0
541,Dolphin,1.0
542,Lego,12.0
543,Fox,11.0
544,Tux,2.0
545,Whale,2.0
546,Marbles,4.0
547,Whale,22.0
548,Dolphin,43.0
549,Marbles,3.0
550,Tux,43.0
551,Marbles,10.0
552,Tux,11.0
553,Fox,10.0
554,Tux,4.0
555,Fox,5.0
556,Monopoly,1.0
557,Tux,22.0
558,Fox,43.0
559,Fox,43.0
```

```
540,Dolphin,2.0
541,Dolphin,1.0
542,Lego,12.0
543,Fox,11.0
544,Tux,2.0
545,Whale,2.0
546,Marbles,4.0
547,Whale,22.0
548,Dolphin,43.0
549,Marbles,3.0
550,Tux,43.0
551,Marbles,10.0
552,Tux,11.0
553,Fox,10.0
554,Tux,4.0
555,Fox,5.0
556,Monopoly,1.0
557,Tux,22.0
558,Fox,43.0
559,Fox,43.0
"order3.csv" [dos] 560L, 8913C
```

```
540,Dolphin,2.0
541,Dolphin,1.0
542,Lego,12.0
543,Fox,11.0
544,Tux,2.0
545,Whale,2.0
546,Marbles,4.0
547,Whale,22.0
548,Dolphin,43.0
549,Marbles,3.0
550,Tux,43.0
551,Marbles,10.0
552,Tux,11.0
553,Fox,10.0
554,Tux,4.0
555,Fox,5.0
556,Monopoly,1.0
557,Tux,22.0
558,Fox,43.0
559,Fox,43.0
"order2.csv" [dos] 560
```

```
Request: 499
Toy Lookup Response:  {'data': {'name': 'Python', 'price': 22.1, 'quantity': 1}}
Query TOY: Marbles
Toy URL: http://ec2-54-82-245-25.compute-1.amazonaws.com:4773/product/Marbles
Toy Lookup Response:  {'data': {'name': 'Marbles', 'price': 36.78, 'quantity': 2}}
Order request: {'name': 'Marbles', 'quantity': 9}
Buy Request Response:  {'error': {'code': 400, 'message': 'Sorry Marbles is out of stock or invalid quantity'}}
Total lookup request:  500
Average lookup request latency: 0.08462022972106933
Total order request:  176
Average order request latency: 0.15861398794434287
Total lookup request:  500
Average lookup request latency: 0.0857695631980896
Total order request:  169
Average order request latency: 0.16417625105592626
Total lookup request:  500
Average lookup request latency: 0.08549097156524658
Total order request:  185
Average order request latency: 0.1551346559782286
Total lookup request:  500
Average lookup request latency: 0.08499780035018921
Total order request:  180
Average order request latency: 0.16275793181525336
Total lookup request:  500
Average lookup request latency: 0.08468157958984375
Total order request:  190
Average order request latency: 0.16096069812774658
```

We conducted several simulations of crash failures on both local machines and AWS to evaluate the resilience of our application. Specifically, we randomly killed a replica in the order server, including the leader, and observed the application's behavior. Remarkably, we found that the application continued to function normally after the replica was killed. Our findings indicate that the application runs as usual, and the client does not notice the crash failures either during order requests or the final order checking phase. Furthermore, when the order server was restored, it automatically synchronized with the databases of other replicas, ensuring that the order database files (order1/2/3.csv) were identical when the client exited. We also verified that the order information retrieved from the database was consistent with the client's locally stored information. Based on these findings, we can confidently conclude that our application is robust to crash failures, and clients will not experience any disruptions.