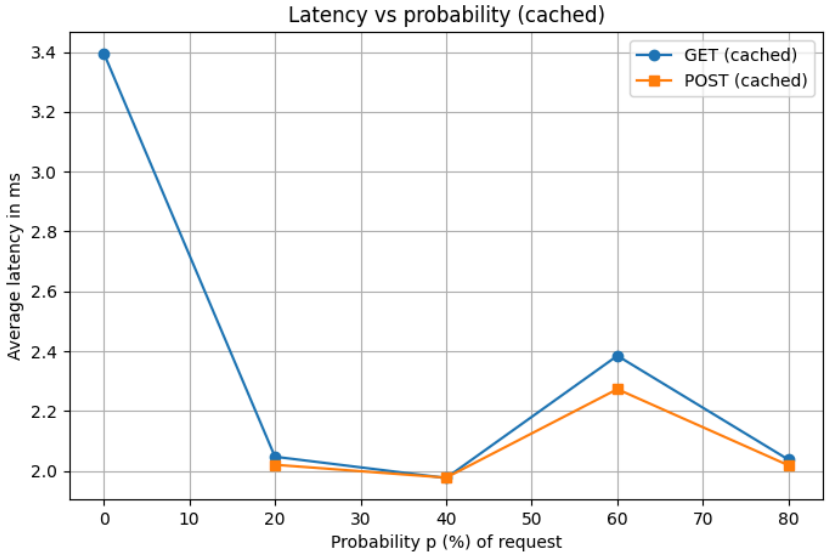
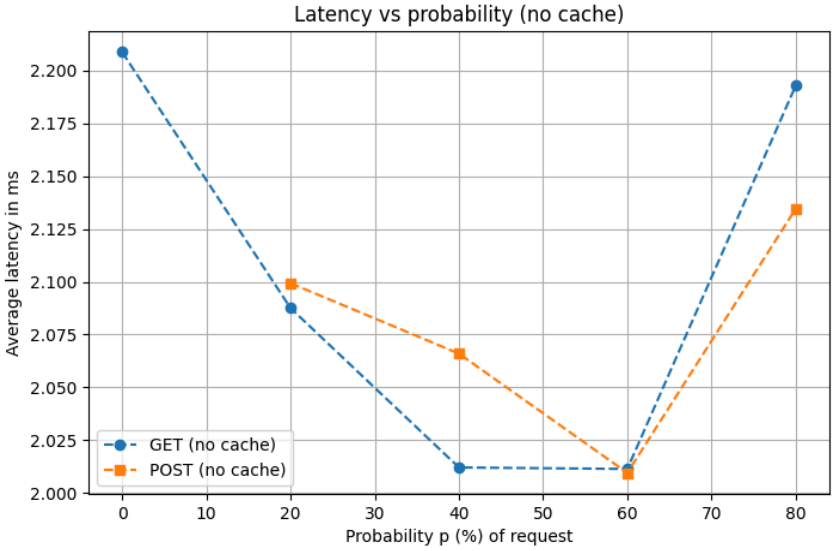
Testing and Evaluation:

1. **estimate how much benefits does caching provide by comparing the results:**





To evaluate the performance benefits of caching, I conducted latency experiments with and without caching enabled, by changing probabilities p of trade requests.

From the results we can see that caching provides most improvements when p = 20%, where average GET latency dropped from 3.18 ms to 2.83 ms (about 11% improvement) and POST latency dropped from 3.19 ms to 2.82 ms (about 11.6% improvement).

For higher values of p (60–80%), the improvements were much lesser (1–4%). This is because of the increase in write-heavy operations, which skips the cache.

At p = 0%, caching does not provide any benefit because there was no cache reuse.

Overall caching significantly improves read performance for moderate workload of read or write. It is less useful when writes dominate or if there is no reuse of cache entries.

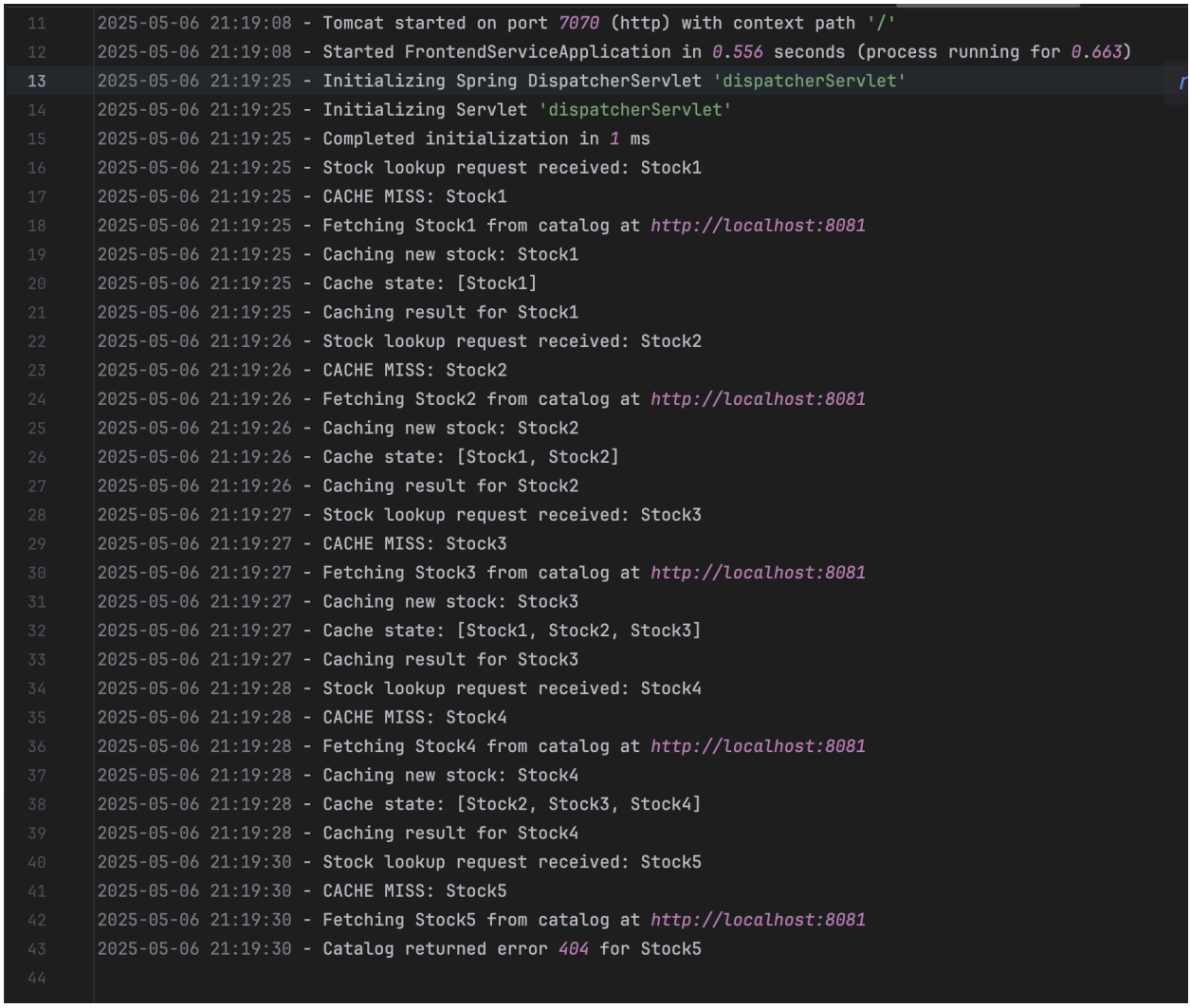
2. **Cache replacement in action:**

   I configured the frontend cache with a size of 3. Then I queried 5 different stocks, the frontend logs showed whether it resulted in a cache hit or miss, and displayed the current cache state. Older entries were being replaced when the limit was exceeded. For eg, after querying Stock1, 2, 3, and then 4, Stock1 was replaced as per the LRU policy as shown in the below screenshots.

test_eval_cache_replacement.sh :

```
#!/bin/bash

# here cache size is 3

echo "testing cache replacement in action"

stock_names=("Stock1" "Stock2" "Stock3" "Stock4" "Stock5")

for stock in "${stock_names[@]}"
do
    echo "querying $stock"
    curl -s http://localhost:7070/stocks/$stock > /dev/null
    sleep 1
done

echo "frontend.log will have all the logs"
```

frontend.log:



3. crash failures and recovery testing:

Steps I followed:

1) Started all the services

```
mvn spring-boot:run -pl src/catalog-service

mvn spring-boot:run -pl src/frontend-service
```

Then in separate terminals

```
mvn spring-boot:run -pl src/order-service -Dspring-boot.run.profiles=replica1

mvn spring-boot:run -pl src/order-service -Dspring-boot.run.profiles=replica2
```

```
mvn spring-boot:run -pl src/order-service -Dspring-boot.run.profiles=replica3
```

2) Started test_client_simulator.sh

```
#!/bin/bash

for i in {1..10}; do
    stock="Stock$((RANDOM % 6 + 1))"
    quantity=$((RANDOM % 5 + 1))
    if (( RANDOM % 2 == 0 )); then
      type="buy"
    else
      type="sell"
    fi

    echo "sending order: $type $quantity of $stock"
    curl -s -X POST http://localhost:7070/orders \
        -H "Content-Type: application/json" \
        -d "{\"name\":\"$stock\",\"type\":\"$type\",\"quantity\":$quantity}" \
        > /dev/null
    sleep 1
done
```

```
(base) lavanika@Lavanikas-MacBook-Pro client % ./test_client_simulator.sh
sending order: buy 3 of Stock1
sending order: sell 5 of Stock4
sending order: sell 5 of Stock3
sending order: sell 4 of Stock6
sending order: sell 3 of Stock2
sending order: sell 5 of Stock1
sending order: buy 3 of Stock3
sending order: buy 4 of Stock5
sending order: sell 1 of Stock3
sending order: sell 5 of Stock6
(base) lavanika@Lavanikas-MacBook-Pro client %
```

3) While running the above script, do ctrl+c to stop one of the replicas (replica3)
4) frontend logs:

```
2025-05-06 22:06:36 - Completed initialization in 1 ms
2025-05-06 22:06:36 - Forwarding order to order service: {name=Stock1, type=buy, quantity=3}
2025-05-06 22:06:36 - Received ping from http://localhost:9091 with replicaId 1
2025-05-06 22:06:36 - Received ping from http://localhost:9092 with replicaId 2
2025-05-06 22:06:36 - Received ping from http://localhost:9093 with replicaId 3
2025-05-06 22:06:36 - Leader selected: http://localhost:9093
2025-05-06 22:06:38 - Forwarding order to order service: {name=Stock4, type=sell, quantity=5}
2025-05-06 22:06:39 - Forwarding order to order service: {name=Stock3, type=sell, quantity=5}
2025-05-06 22:06:40 - Forwarding order to order service: {name=Stock6, type=sell, quantity=4}
2025-05-06 22:06:41 - Forwarding order to order service: {name=Stock2, type=sell, quantity=3}
2025-05-06 22:06:42 - Forwarding order to order service: {name=Stock1, type=sell, quantity=5}
2025-05-06 22:06:43 - Forwarding order to order service: {name=Stock3, type=buy, quantity=3}
2025-05-06 22:06:44 - Forwarding order to order service: {name=Stock5, type=buy, quantity=4}
2025-05-06 22:06:45 - Forwarding order to order service: {name=Stock3, type=sell, quantity=1}
2025-05-06 22:06:45 - current leader http://localhost:9093 is unreachable, re-electing leader
2025-05-06 22:06:45 - Received ping from http://localhost:9091 with replicaId 1
2025-05-06 22:06:45 - Received ping from http://localhost:9092 with replicaId 2
2025-05-06 22:06:45 - Could not reach replica at http://localhost:9093: I/O error on GET request for "http://localhost:9
2025-05-06 22:06:45 - Leader selected: http://localhost:9092
2025-05-06 22:06:46 - Forwarding order to order service: {name=Stock6, type=sell, quantity=5}
```

5) After restarting the replica3, we can verify that all the replicas are synced.

```
curl http://localhost:9091/orders/10

curl http://localhost:9092/orders/10

curl http://localhost:9093/orders/10
```

Output:

```
(base) lavanika@Lavanikas-MacBook-Pro client % curl http://localhost:9091/orders/1
{"data":{"name":"Stock3","type":"sell","quantity":1,"number":1}}%
(base) lavanika@Lavanikas-MacBook-Pro client % curl http://localhost:9092/orders/1
{"data":{"name":"Stock3","type":"sell","quantity":1,"number":1}}%
(base) lavanika@Lavanikas-MacBook-Pro client % curl http://localhost:9093/orders/1
{"data":{"name":"Stock3","type":"sell","quantity":1,"number":1}}%
(base) lavanika@Lavanikas-MacBook-Pro client %
```

From the above, we can see that during the crash of the leader (replica3), the frontend re routes requests to the next highest replica (replica2). Client did not experience errors, and this shows fault transparency.

Once the crashed replica was restarted, it queried the current leader and synced missing orders.
I queried the same order ID (/orders/1) from all replicas and received the same data.