Flask API Performance Test Report

1. Overview of the API

This project implements a basic RESTful API using Python and Flask. The API provides the following endpoints:
- GET /items: Returns a list of TODO items (stored in memory).
- POST /items: Adds a new item to the in-memory list.

2. Testing Methodology & Tools Used

The API was tested using the following tools and methods:
- curl: For adding and retrieving 50 TODO items to simulate real usage.
- curl -w: To measure response time.
  Command used: `curl -w "\nTime: %{time_total}s\n" http://localhost:5000/items`
- ApacheBench (ab):
  - Load Test: `ab -n 1000 -c 100 http://127.0.0.1:5000/items`
  - Stress Test: `ab -n 5000 -c 200 http://127.0.0.1:5000/items`
- Windows Task Manager: For monitoring system resource usage during 100 concurrent requests.

3. Observations

- Average response time for GET /items (with 50 items):
  - Without Docker (run localy): Time: 0.216508s
  - Using Docker: Time: 0.005146s
- Reason: Docker provides a lightweight and isolated environment, reducing system overhead and improving response time compared to the local machine.
- Load Test Results:
  - 100 concurrent requests completed with no errors.
  - Suitable for light to medium traffic.

4. System Monitoring (During Load Test)

- CPU Usage: ~0.3% → ~12%
- Memory Usage: ~22 MB → ~23.3 MB
- No memory leaks or abnormal spikes were observed.

5. Conclusion

The API performed well under light-to-medium load (100 concurrent users), with no failed requests. It is reliable for basic usage scenarios and demonstrates efficient resource usage.

6. Improvements for Increased Traffic

To handle increased traffic, I would implement **caching** for frequently accessed data to reduce database or memory operations on repeated requests

I recommend using **Redis** for caching due to its speed and efficiency in handling frequent read operations.

8. Bonus Ideas

1. Stress Test

I increased the number of requests and concurrency level

Using ApacheBench with 5000 requests and 200 concurrent requests:
- Time taken: 14.587 seconds
- Complete requests: 5000
- Failed requests: 0
- Longest request: 14527ms

→ Even under heavy load, the API remained stable.

2. SQLite Advantage

Switching to SQLite does not improve single-request performance but adds persistent storage and better scalability.

It handles large datasets and concurrent access more reliably and avoids data loss on server restart.

To test: run `python sql-lite-test.py` using the same tools.

9. Attachments

I attached screenshots of the tests I performed, including response time measurements, load test results, and system resource usage during load testing