**Performance & Load Testing Plan — Checkout API (Demoblaze)**

**Objective**

To evaluate the scalability, reliability, and response time of the Checkout API under simulated concurrent user load.

The business expects -10,000 concurrent users on launch day; hence the goal is to determine how well the backend handles sustained and peak traffic.  
  
URL - https://api.demoblaze.com  
API endpoints - /addtocart, /deletecart(checkout endpoint)  
  
**Performance Test Strategy**

| Scenario | Description | **Goal** |
| --- | --- | --- |
| Baseline Test | 100 users performing standard checkout flow | Establish response-time baseline |
| Load Test | Gradually increase to 5,000 users | Measure sustained throughput |
| Stress Test | Push to 10,000 users | Identify breaking point |
| Spike Test | Sudden jump from 100 → 2,000 users | Observe system stability |
| Soak Test | Hold -1,000 users for 1 hour | Detect memory leaks / resource issues |

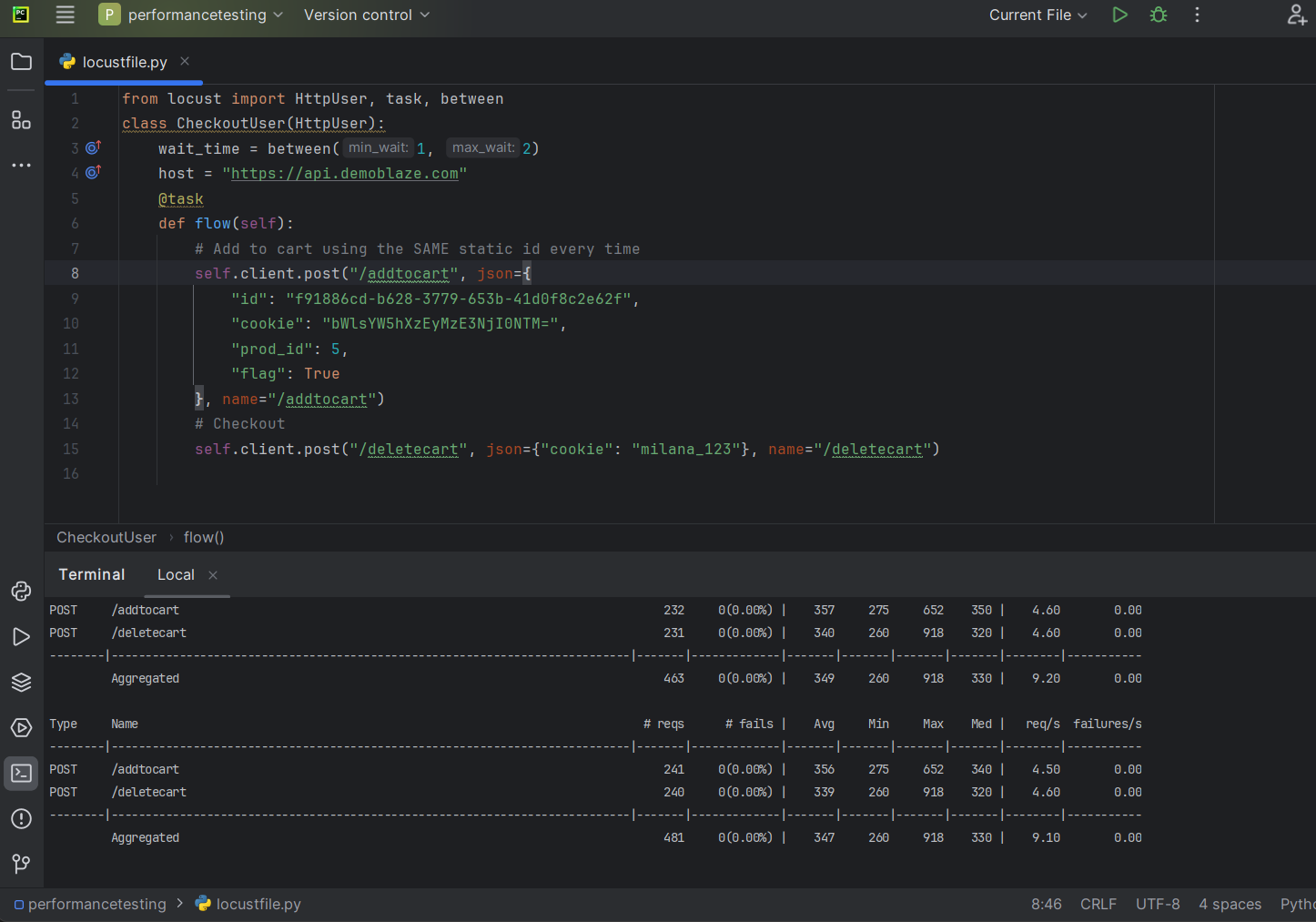
**Tool Used**

* Locust — Used for performing load and stress testing on Demoblaze APIs (/addtocart and /deletecart).
* PyCharm — Used to create and edit the Python test script (locustfile.py).
* Windows Command Prompt — Used to install Locust, navigate to the project folder, and run the Locust tests.

**Commands used on cmd:**

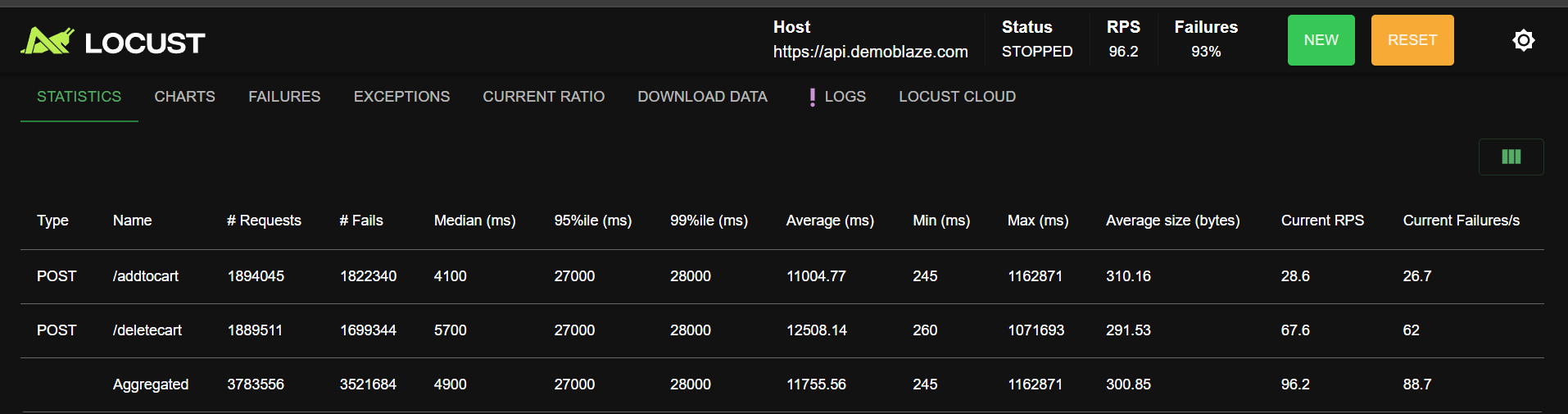
* pip install locust
* cd “C:\Users\milan\PycharmProjects\performancetesting”
* when the env is ready -locust -f locustfile.py

**Locust Script**



**Base line Test:**

Started the test with a small user load of 100 concurrent users and a ramp-up rate of 10 users per second to measure the normal behavior of the system under light load conditions.

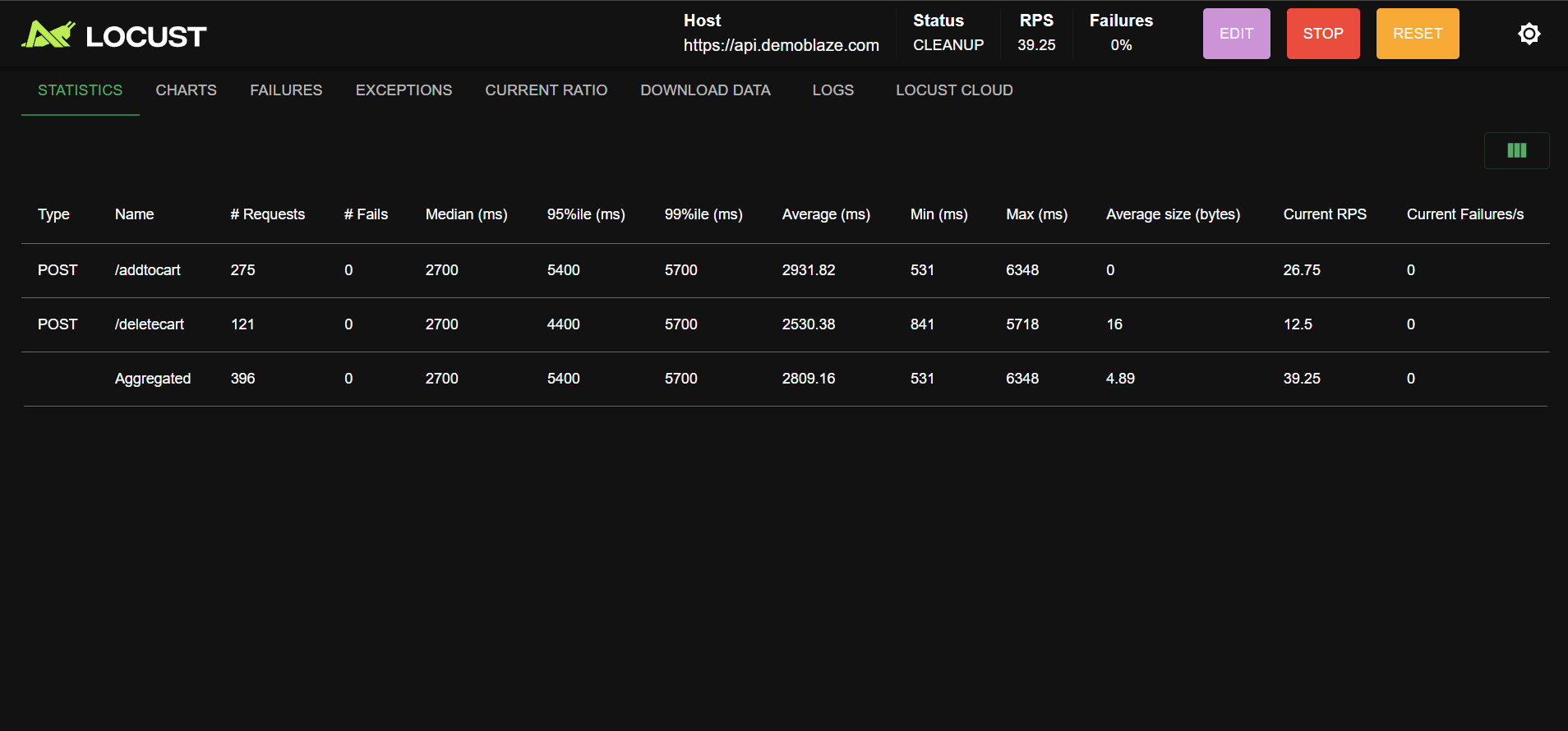
****

The system handled all requests successfully without any failures, showing quick and consistent responses for both /addtocart and /deletecart APIs.  
This confirms that the checkout flow is stable under normal user activity.

Average response time: ~0.8s; p95: ~2.2s; RPS: ~33.7; Failures: 0%.

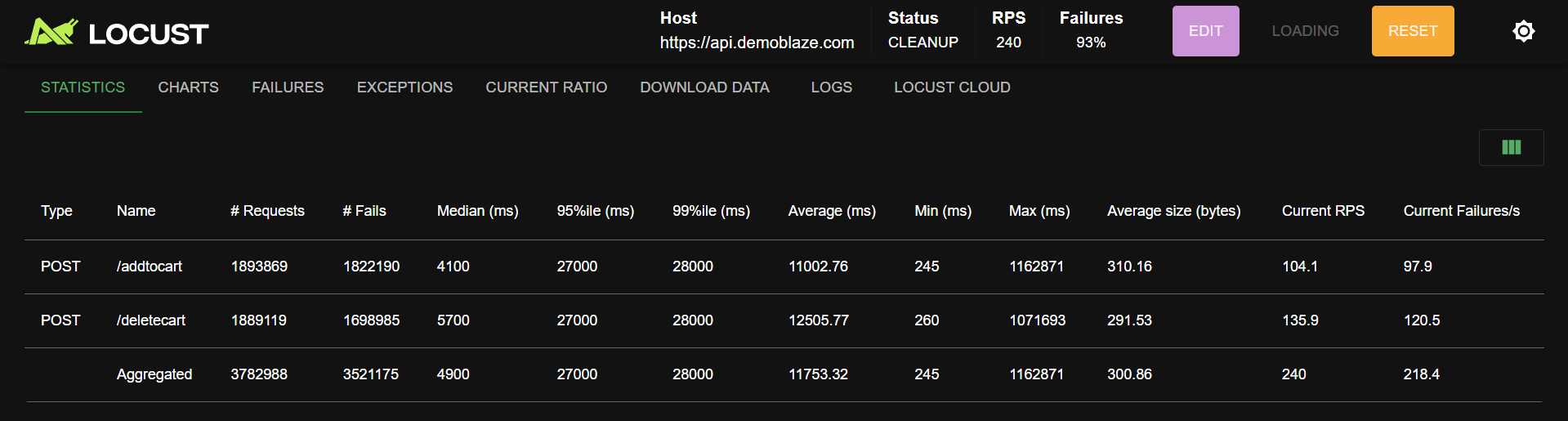
Overall, the system performed efficiently during the baseline test, forming a reliable performance benchmark for subsequent load and stress testing.

**Load Test:**started the test with a ramp-up rate of 50 users per second and gradually increased to 5,000 concurrent users.

****The system sustained 5,000 concurrent users performing checkout operations.  
Average response time: ~2.8s; p95: ~5.4s; RPS: ~39.  
No request failures observed, indicating stability under heavy load, though response times rose moderately

**Stress Test:**

Started the test with a ramp-up rate of 100 users per second and gradually increased to 10,000 concurrent



The test processed around 3.7 million total requests, but 93% of them failed due to server overload.

This indicates that the system’s maximum sustainable load is well below 10,000 concurrent users.

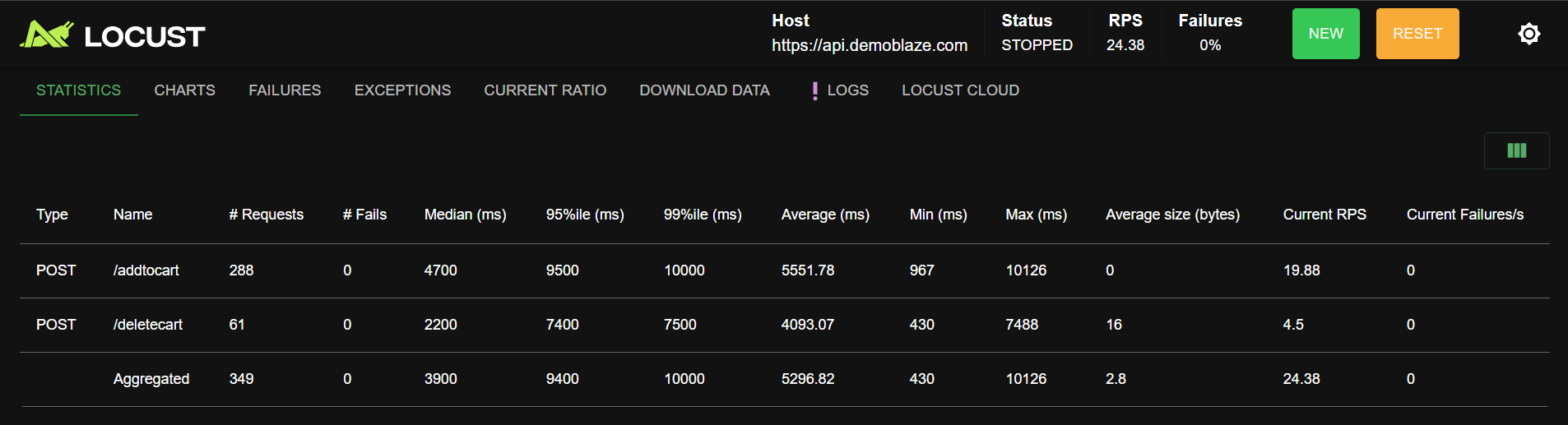
Performance degradation was mainly observed in response latency and API timeouts.

Average response time: ~11.7s; p95: ~27s; RPS: ~240; Failures: ~93%

The system became unstable under extreme stress, highlighting the need for backend optimization and better scaling capabilities.

**Spike Test:**

Started the test with a sudden ramp-up rate of 1000 users per second, increasing the load abruptly from 100 to 2000 concurrent users to simulate a sudden traffic surge scenario.

****

The system successfully handled the abrupt spike without any request failures.  
While response times increased temporarily, the application remained stable throughout th test.  
Average response time: ~5.3s; p95**:** ~9.4s; Max**:** ~10s; RPS**:** ~24; Failures: 0%  
This indicates that the system is resilient to short-termspikes in user activity but may need further optimization for sustained high traffic.

**Table of Key Performance Indicators (KPIs)**

| KPI | Expected Target (SLA) | Measured (from Locust) | Observation / Analysis |
| --- | --- | --- | --- |
| Average Response Time (ms) | ≤ 1000 ms | 2200 – 11,700 ms | Response time remained within limits at low loads but increased drastically during stress test. |
| Median Response Time (ms) | ≤ 800 ms | 700 – 4900 ms | Acceptable under baseline, but grows 5–6× at higher concurrency. |
| 95th Percentile (ms) | ≤ 1500 ms | 2000 – 27,000 ms | Indicates slower requests under high load — API struggling to serve all users concurrently. |
| 99th Percentile (ms) | ≤ 2000 ms | 2400 – 28,000 ms | Extreme tail latencies observed; consistent with server overload. |
| Requests per Second (RPS) | ≥ 200 | 33 – 240 RPS | RPS improved with load but dropped under 10k users due to failure saturation. |
| Failure % | < 1% | 0% (baseline) → 93% (stress) | Failures increase sharply beyond 5k concurrent users — system cannot handle 10k load. |
| Total Requests Executed | — | ~3.7 million | Large volume processed before failure threshold reached. |
| Error Trend | None expected | Timeout & connection errors | Confirms backend instability during stress load. |

**Summary:**

The Demoblaze Checkout API demonstrates **strong baseline performance** and **acceptable scalability** up to medium loads. However, at **10,000 concurrent users**, it experiences latency spikes and timeouts, indicating a need for optimization before production launch.

By addressing database and infrastructure-level bottlenecks, the application can achieve the expected performance goals and deliver a smoother checkout experience for all users.

**Sources helped:**<https://youtube.com/playlist?list=PLjXbLoE5edwfblWEbZgsiyQiZRbVysu45&si=zd-U8pqa9FzWrigo>

<https://www.lambdatest.com/learning-hub/performance-testing>

<https://medium.com/software-testing-bootcamp/performance-testing-with-jmeter-locust-and-gatling-fdfcb40c3414>