Assignment 8

Report

Report completed by Adam Satyshev

IT-2302

Test Environment

* Operative System: Windows 11 Pro x64, Version 24H2, Build 26100.3194
* Device: Laptop Legion 5 15IAH7H (Desktop)

Objectives and Scope

The goal of this performance testing is to assess the behavior and responsiveness of a sample web API under various load conditions, browsers, and network settings. This testing identifies potential bottlenecks and provides actionable strategies to improve performance.

Strategies and Methods

Locust was used to simulate user behavior and analyze server response. Chrome, Firefox, and Edge browsers were used for browser-specific load analysis. The test included analysis under simulated different network speeds.

1. Performance Metrics Analysis

Key Metrics Summary:

|  |  |
| --- | --- |
| Metric | Value |
| Total Requests | 86 |
| Total Failures | 64 (74.42%) |
| Avg Response Time (All) | 206 ms |
| Max Response Time (All) | 480 ms |
| Median (50%) Response Time | 180 ms |
| Throughput | 4.54 req/s |
| TTFB (Chrome) | 223 ms |
| TTFB (Firefox) | 288 ms |
| TTFB (Edge) | 192 ms |
| Browser Load Times (avg) | ~3.78 s |

Percentile Breakdown (Aggregated):

50% (Median): 180 ms – Majority of responses are decent.  
90%: 370 ms – Acceptable, but nearing user experience limits.  
99–100%: 480 ms – Long tail with some slow outliers.

Error Rate:

POST /api/users: 42 requests – 100% failure  
Overall failure rate: 74.42%, which is critical.

Identified Bottlenecks and Issues

* High Failure Rate (POST requests): All POST requests failed – suggests either:  
  - Backend API is not handling POST logic correctly.  
  - Data sent might be malformed or server rejected it.
* High Load Time in Browsers: Load times across browsers are ~3.85s, suggesting frontend optimization issues (scripts, images, CSS not optimized).
* Response Time Peaks: While average is acceptable, max times of 480 ms indicate backend processing bottlenecks or inconsistent response handling under load.
* Low Throughput (4.54 req/s): May indicate blocking calls, synchronous processing, or lack of concurrency handling.

Proposed Performance Optimization Strategies

1. Backend Improvements

* Fix POST failure issue:  
  - Check API schema validation, input sanitization, and DB connectivity.  
  - Log error messages to trace failures.
* Optimize database interactions:  
  - Index critical fields.  
  - Use bulk inserts where applicable.
* Enable Caching:  
  - Cache frequent GET responses (e.g., Redis or in-memory).

1. Frontend Optimization

* Reduce load time:  
  - Minify JS/CSS.  
  - Defer or lazy-load non-critical resources.  
  - Optimize image formats and compression.
* Improve TTFB:  
  - Reduce server-side processing delays.  
  - Use faster hosting/CDN for static files.

1. Load Handling

* Use async processing (e.g., Celery, async/await) for non-blocking operations.
* Scale backend horizontally or enable autoscaling to handle sudden surges.

1. Locust Testing Enhancements

* Increase test duration to capture sustained performance.
* Test under different user patterns: ramp-up/down, spike testing, soak testing.
* Include authentication, varying payloads, and error scenarios.

Conclusion

The current performance is unstable, with:  
- Unacceptable error rate (74%),  
- Moderate response times under load,  
- High browser load times (~3.8s), and  
- Low throughput.  
  
Priority Action: Fix failing POST endpoint and optimize backend logic under concurrency.