PERFORMANCE TEST REPORT - PETSTORE API (RELEVANT ENDPOINTS: STORE AND USERS)

Andres Silva
andresdavidsilva@hotmail.com

2025

Introduction
This report outlines the results of the performance tests conducted on the key endpoints of the Petstore API, accessible through the Swagger UI for the Petstore API. The tests focused on the most critical endpoints for store operations: pet, store, and users.

Testing Objectives
The primary objective of the tests was to evaluate how the pet, store, and users endpoints handle different traffic volumes and respond under constant load, traffic spikes, and stress conditions. The specific objectives were:

- Assess the ability of the pet, store, and users endpoints to handle a steady user load.
- Measure the performance and stability of these endpoints under peak traffic conditions.
- Determine the breaking point of the endpoints when subjected to a stress test.
- Identify any response time degradation or errors under various conditions.

Scope of Testing
The following critical test cases were identified for automation and performance testing:

- User Management: Creating and logging in users through the API.
- Pet Management: Adding, updating, and retrieving pet information.
- Store Transactions: Placing orders, retrieving inventory, and managing orders.

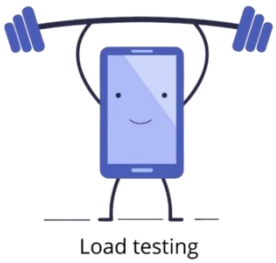
Each of these functionalities plays a key role in ensuring the API operates reliably under functional and performance testing scenarios.

- GET /pet/{id}: Find Pets by ID.
- GET /store/inventory: Get Inventory.
- POST /store/order: Place an order for a pet.
- POST /user: Create a user.
- GET /user/login: Logs user into the system

Selected Performance Testing Type

To ensure optimal API performance, the following three types of performance tests were chosen:

- Purpose:
  Push the system beyond its expected limits to determine its breaking point.
- Justification:
  Identifies how the API degrades under extreme conditions and whether it recovers gracefully.
- Test Parameters:
  80-100 concurrent users simulating peak traffic.
- Success Criteria:
  Response times should ideally stay <0.5 seconds, with an error rate below 2% (http_req_failed < 0.02).



- Purpose:
  Simulate expected real-world usage by applying an average number of concurrent users performing standard operations.
- Justification:
  Ensures the system can handle normal traffic loads efficiently.
- Test Parameters:
  10-40 concurrent users executing typical transactions.
- Success Criteria:
  99% of requests should complete within <0.5 seconds (p (99) < 500ms).

- Purpose:
  Assess the system's ability to handle increasing user loads over time.
- Justification:
  Ensures the API can efficiently scale with gradual traffic growth.
- Test Parameters:
  Load increases gradually from 10 to 200 users over a defined period.
- Success Criteria:
  Response times should remain stable, and recovery time after a spike should be within an acceptable threshold.

Expected Metrics Results

These values were defined based on typical API usage patterns and industry benchmarks. To accurately simulate real-world conditions, we established the following load parameters:

- Response Time: Acceptable range between 100-500ms under normal conditions.
- Average Load: Simulating 10-40 concurrent users performing standard operations.
- Peak Load: Testing with 80-100 concurrent users to observe system behavior under high traffic.

Acceptance Criteria

To evaluate test success, the following performance benchmarks were established:

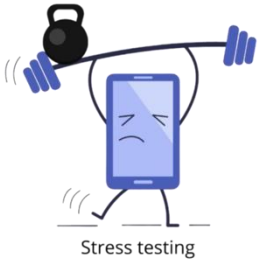
- Response Time:
  - Under normal load, 99% of requests should complete within <0.5 seconds (p (99) < 500ms).
  - Under stress conditions, system degradation should be graceful, with response times ideally below 0.5 seconds.
- Error Rate:
  - Normal load: Less than 1% (http_req_failed < 0.01).
  - Stress conditions: Should not exceed a 2% failure rate (http_req_failed < 0.02).
- Scalability:
  - The system should maintain stable response times as user load increases gradually.
  - For peak loads, recovery time after a spike should be within an acceptable threshold.

These thresholds ensure that the API meets industry standards for responsiveness and reliability.

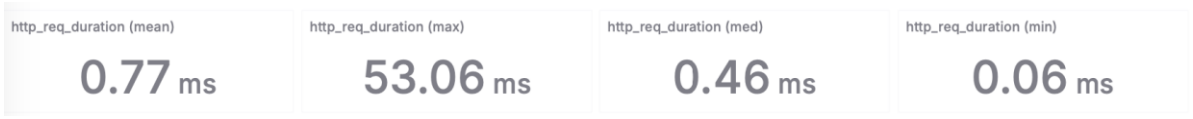
Test cases

Test Case	Test Type	Description	Parameters	Expected Outcome
`stress_test find_pets_by_status`	Stress Test	Test retrieving pets by status under high traffic load	80-100 concurrent users, peak traffic	Response < 0.5s, error rate < 2%
`stress_test get_inventory`	Stress Test	Test getting inventory under high traffic conditions	80-100 concurrent users, peak traffic	Response < 0.5s, error rate < 2%
`stress_test place_order`	Stress Test	Test placing orders under high traffic conditions	80-100 concurrent users, peak traffic	Response < 0.5s, error rate < 2%
`load_test find_pets_by_status`	Load Test	Test retrieving pets by status with normal traffic load	10-40 concurrent users	Response < 0.5s, 99% of requests successful
`load_test add_pet`	Load Test	Test adding a new pet to the store under normal load	10-40 concurrent users	Response < 0.5s, 99% of requests successful
`load_test place_order`	Load Test	Test placing orders under normal load	10-40 concurrent users	Response < 0.5s, 99% of requests successful
`load_test create_user`	Load Test	Test user creation functionality under normal load	10-40 concurrent users	Response < 0.5s, 99% of requests successful
`scalability_test find_pets_by_status`	Scalability Test	Test scalability of finding pets by status as load increases	10-200 concurrent users, gradual increase	Stable response times as load increases
`scalability_test get_inventory`	Scalability Test	Test scalability of getting inventory as load increases	10-200 concurrent users, gradual increase	Stable response times as load increases
`scalability_test place_order`	Scalability Test	Test scalability of placing orders as load increases	10-200 concurrent users, gradual increase	Stable response times as load increases
`scalability_test create_user`	Scalability Test	Test scalability of user creation as load increases	10-200 concurrent users, gradual increase	Stable response times as load increases

Key Results



- Response times should ideally stay <0.5 seconds. ☑



find_pets_by_status

get_inventory

place_order

- Should not exceed a 2% failure rate (http_req_failed < 0.02). ☑

place_order

get_inventory

find_pets_by_status

- The system handled up to 100 virtual users without failures, except for 0,8% to place_order. ✅
- RPS remained stable (~47.70 RPS). ✅

Key Results

- 9% of requests should complete within <0.5 seconds (p (99) < 500ms). ❌
- Less than 1% (http_req_failed < 0.01). ❌ to:
  - add_pet

  - place_order

- System sustained ~18.50 RPS under load. ✅

Key Results

- Response times should remain stable and recover after load spikes. ✅ (For get_inventory only).
- Create_user and find_pets_by_status completely failed (100% failure rate). ❌

- System handled 200 VUs on get_inventory and place_order with minimal failures. ✅
- Create_user and find_pets_by_status need urgent fixes to handle scalability. ❌

Detailed Test Results by Endpoint

Test/Endpoint	Status	Success Rate	Failure Rate	Avg Response Time	Max Response Time	TPS	RPS	Concurrency (VUs)
Stress Test - find_pets_by_status	✅ Success	100%	0%	774.38µs	53.05ms	47.72	47.72	100
Stress Test - get_inventory	✅ Success	100%	0%	787.73µs	714.97ms	47.70	47.70	100
Stress Test - place_order	⚠️ Partial Failures	99%	0.80%	757.51µs	77.79ms	47.69	47.69	100
Load Test - find_pets_by_status	✅ Success	100%	0%	1.17ms	258.67ms	18.50	18.50	40
Load Test - add_pet	❌ Critical Failure	8%	91.73%	1.92ms	47.54ms	18.49	18.49	40
Load Test - place_order	⚠️ Moderate Failures	98%	1.14%	1.4ms	201.32ms	18.52	18.52	40
Load Test - create_user	❌ Severe Failure	28%	71.29%	1.54ms	39.33ms	18.48	18.48	40
Scalability Test - find_pets_by_status	❌ Critical Failure	0%	100%	483.02µs	23.04ms	81.87	81.87	200
Scalability Test - get_inventory	✅ Success	100%	0%	629.06µs	52.7ms	82.00	82.00	200
Scalability Test - place_order	⚠️ Moderate Failures	99%	0.55%	625.86µs	47.82ms	81.88	81.88	200
Scalability Test - create_user	❌ Critical Failure	0%	100%	517.08µs	69.6ms	81.72	81.72	200

Analysis & Recommendations

- How Many Users Scaled and for How Long?
    - Stress Tests scaled up to 100 virtual users for 37 minutes, and the system handled it well except for place_order (0.8% failures).
    - Load Tests sustained 40 virtual users for 20 minutes but *add_pet* and *create_user* showed severe failures (>70% failure rate).
    - Scalability Tests reached 200 virtual users for 38 minutes and create_user and *find_pets_by_status* completely failed (100% failure rate).
- Bottlenecks in *create_user* and *add_pet*
    - Cause: Too many simultaneous connections overwhelming the API.
    - Posible solution: Implement caching, adjust connection limits, and optimize queries.
- Failures in *place_order* under high load
    - Cause: Database transaction locks.
    - Posible solution: Optimize lock handling or improve system scalability.

- High Response Times in get_inventory
    - Cause: Slow database queries.
    - Posible solution: Index database and add caching.

Conclusions

The stress tests successfully evaluated the system's capacity under 100 virtual users (VUs) for 37 minutes. Most endpoints, including *find_pets_by_status* and get_inventory, performed within acceptable limits. However, place_order showed a 0.8% failure rate, indicating that database transactions might be causing contention issues.

Key Observations:

- Response times remained stable overall (avg ~0.77ms).
- Failures were minimal but indicate a potential limit in order processing capacity.
- No significant degradation was observed in system throughput (TPS/RPS remained stable at ~47.7 requests per second).

The load testing revealed major performance issues in *add_pet* and *create_user*, with 91.73% and 71.29% failure rates, respectively. *place_order* also exhibited moderate failures (1.14%), suggesting that the system struggles with handling concurrent requests efficiently under sustained traffic.

Key Observations:

- *add_pet* and create_user encountered high failure rates due to backend constraints.
- *place_order* struggled with request handling under prolonged load.
- Throughput decreased (TPS/RPS dropped to ~18.5 requests per second compared to 47.7 in **stress** tests).

The scalability tests assessed the system’s ability to handle 200 virtual users (VUs) for 38 minutes, exposing critical weaknesses in *create_user* and *find_pets_by_status*, which completely failed (100% failure rate). Only *get_inventory* scaled successfully.

Key Observations:

- *create_user* and *find_pets_by_status* failed under 200 VUs, indicating that these APIs are not designed to scale effectively.

- *place_order* managed to handle 200 VUs with a 0.55% failure rate, which is still within an acceptable margin but requires optimization.
- API throughput reached 81.87 TPS/RPS, which suggests the system can handle high request rates if properly optimized.**:**

Recommendations

- Optimize database transaction management to avoid bottlenecks under peak load.
- Implement database connection pooling to improve concurrent request handling.
- Monitor API throughput during peak traffic to ensure response times remain within an acceptable range.
- Re-run the stress test after optimizations to validate performance improvements.
- Implement rate limiting and load balancing mechanisms to ensure better API performance.
- Investigate thread pool exhaustion and database connection limits in *create_user*.
- Optimize backend processes to handle concurrent database writes efficiently.
- Consider adding horizontal scaling capabilities to support larger traffic loads.
- Perform another scalability test after implementing optimizations to verify improvements.
- Improve backend resource allocation to avoid excessive failures under sustained load.
- Optimize validation logic and request processing efficiency for *add_pet* and *create_user*.
- Review system architecture for potential bottlenecks in database writes and API throttling.
- Introduce better error handling and request retry mechanisms to improve resilience.
- Conduct additional load tests after optimization to evaluate infrastructure improvements.