Load Testing Test Case Document  
Project: Performance Testing for Simplita.ai  
Prepared By: Senior Performance Tester (12+ Years Experience)  
Date: [Insert Date]  
Version: 1.1

1. Test Overview

* Objective: Assess the performance, stability, and scalability of https://app.simplita.ai/ under varying load conditions.
* Scope:
  + Load testing on critical workflows including authentication, dashboard access, transactions, API responses, and concurrent user handling.
  + Measure system limits and identify potential bottlenecks.
* Assumptions:
  + A stable pre-production/staging environment is available.
  + Performance monitoring tools are integrated.
  + Network latency is within acceptable limits.

2. Test Environment

* Application URL: https://app.simplita.ai/
* Test Environment: Pre-Production / Staging
* Infrastructure Setup:
  + Web Server: Nginx/Apache
  + Database: PostgreSQL/MySQL
  + App Server: Node.js / Python FastAPI
  + Cloud Provider: AWS / Azure / GCP
  + Hardware Configuration:
    - CPU: 16-Core Processor
    - RAM: 64GB
    - Disk: 1TB SSD
    - Network: 10Gbps bandwidth
* Load Testing Tools: JMeter / Gatling / K6 / Locust / LoadRunner
* Monitoring Tools: Grafana, New Relic, Prometheus, Datadog

3. Test Scenarios & Workflows

3.1. Load Test Cases

| Scenario ID | Test Scenario | Target Load | **Target Time** | **Loop Count** | Expected Response Time |
| --- | --- | --- | --- | --- | --- |
| TC-001 | User Login (Valid Credentials) | 1000 concurrent users |  |  | < 2 sec |
| TC-002 | User Login (Invalid Credentials) | 1000 concurrent users |  |  | < 2 sec |
| TC-003 | Navigate to Dashboard | 2000 concurrent users |  |  | < 3 sec |
| TC-004 | Load Dashboard Widgets | 2000 concurrent users |  |  | < 3 sec |
| TC-005 | Execute Core API Call | 3000 concurrent users |  |  | < 3 sec |
| TC-006 | Log Out | 1000 concurrent users |  |  | < 2 sec |
| TC-007 | Data Fetch from Backend | 5000 concurrent users |  |  | < 3 sec |
| TC-008 | Multi-User Transactions | 5000 concurrent users |  |  | < 5 sec |
| TC-009 | File Upload (5MB) | 1000 concurrent users |  |  | < 4 sec |
| TC-010 | File Download (10MB) | 1000 concurrent users |  |  | < 4 sec |
| TC-011 | Simultaneous API Requests | 10,000 requests/sec |  |  | < 3 sec |
| TC-012 | Stress Test - Peak Load | 50,000 concurrent users |  |  | < 5 sec |
| TC-013 | Scalability Test - Auto Scaling | 100,000 virtual users |  |  | < 4 sec |
| TC-014 | Database Query Performance | 500 concurrent queries |  |  | < 2 sec |
| TC-015 | Long Running Session Handling | 5000 users |  |  | No Session Timeout |

4. Test Execution Plan

* Ramp-Up Strategy:
  + Start with 10 users, then increase by 100 users every 10 seconds until peak load is reached.
* Steady-State Load:
  + Maintain peak load for 1 hour to evaluate stability.
* Ramp-Down Strategy:
  + Gradual user reduction to monitor system recovery.
* Concurrency Levels: 100, 1000, 5000, 10,000, 50,000 users
* Think Time: 2-5 seconds per request

5. Performance Metrics & Benchmarks

| Metric | Acceptable Threshold |
| --- | --- |
| Average Response Time | < 3 sec |
| Peak Response Time | < 5 sec |
| Error Rate | < 1% |
| Throughput | Minimum 500 TPS |
| CPU Utilization | < 75% |
| Memory Utilization | < 80% |
| Database Query Response Time | < 2 sec |
| API Call Failure Rate | < 0.5% |

6. Monitoring & Reporting

* Metrics Monitored:
  + CPU & Memory usage
  + Disk I/O & Network Bandwidth
  + Error Logs & API Latency
  + DB Query Performance
* Reporting Format:
  + Performance Summary Report
  + Charts & Graphs for trends
  + Bottleneck Identification & Recommendations

7. Risks & Mitigation Plan

| **Risk** | **Mitigation Strategy** |
| --- | --- |
| High Response Time | Optimize DB queries, use caching strategies |
| Server Crash | Implement auto-scaling and load balancing |
| High Error Rate | Improve exception handling and logging |
| API Rate Limits Reached | Implement proper API throttling |
| Database Bottlenecks | Optimize indexing, use read replicas |

8. Conclusion & Recommendations

* Ensure the application sustains peak load without performance degradation.
* Optimize API and database responses to minimize latency.
* Implement auto-scaling strategies to handle high traffic.
* Review caching mechanisms for improved response time.
* Continuously monitor system performance with integrated tools.

Approval:  
Performance Test Lead: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Project Manager: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

End of Document