Maogma Loan Management System

Performance and Security Testing Plan

Summary

This plan highlights the objectives, test scenarios, tools and metrics, as well as the execution plan that can be used to evaluate both performance and security of the Maogma Loan Management System.

Performance Testing Plan

1. **Objective:**

Ensure the application can handle a significant number of concurrent operations without degrading performance.

2. **Test Scenarios:**
    a. **Concurrent API Calls:** Simulate multiple users performing various operations (e.g., adding loans, retrieving loan records).
    b. **Database Load Testing:** Measure the database's performance under heavy read and write operations.
    c. **Stress Testing:** Test the application's behaviour under extreme load conditions to determine breaking points.
    d. **Spike Testing:** Introduce sudden spikes in traffic to observe how the application handles unexpected surges.
    e. **Latency Testing:** Measure response times for critical API endpoints under load.
3. **Tools:**
    a. **Postman Collection Runner** for basic load testing.
    b. **Apache JMeter** for advanced testing.
4. **Performance Metrics to Capture:**
    a. Response time.
    b. Throughput (requests per second).
    c. Error rate.
    d. System resource utilization (CPU, memory, and database connections).

5. **Execution Plan:**
    a. Incremental load testing for concurrency
    b. Test scenarios including login, data retrieval, and data creation.
    c. Run tests in both peak and off-peak hours to capture varied results.
6. **Analysis:**
    a. Compare results against SLAs (e.g., response time under 200ms for 95% of requests).
    b. Identify bottlenecks and optimize code, database queries, or server configuration.

Security Testing Plan

1. **SQL Injection Testing:**
    a. Inject malicious SQL queries (' OR 1=1; --) in input fields to test for vulnerabilities.
    b. Use automated tools i.e **SQLMap**.
    c. Ensure proper use of parameterized queries
2. **Cross-Site Scripting (XSS):**
    a. Input malicious scripts (<script>alert('XSS')</script>) into text fields to check if they are executed.
    b. Validate all user inputs on both client and server sides.
    c. Implement proper output encoding.
3. **Authentication and Authorization:**
    a. Test endpoints with invalid, expired, or missing tokens to ensure proper handling.
    b. Verify that users cannot access data or actions they're not authorized for.
    c. Ensure secure password storage using bcrypt or similar hashing algorithms.
4. **Data Validation and Sanitization:**
    a. Verify that inputs are properly sanitized to prevent injection attacks.
    b. Check for enforcement of input length and type constraints.

5. **Session Management:**
    a. Test for secure session cookies (HttpOnly, Secure, SameSite flags).
    b. Ensure sessions are invalidated after logout or inactivity.

6. **Error Handling:**
    a. Test for unhandled errors revealing sensitive information (e.g., stack traces).
    b. Ensure error messages are generic and do not leak implementation details.
7. **Rate Limiting:**
    a. Simulate brute-force attacks on login or other sensitive endpoints.
    b. Verify rate-limiting mechanisms to block excessive requests from the same IP.
8. **Secure Data Transmission:**
    a. Ensure all API calls are made over HTTPS.
    b. Verify the absence of mixed content (e.g., loading resources over HTTP).

Conclusion

This plan if followed will ensure that the loan management system is performance friendly and secure.