Load Testing Strategy

1. Identify Test Objectives:

Determine the maximum number of concurrent users the service should support.

Identify acceptable response times for each endpoint.

Determine the allowable error rate (e.g., <1%).

2. Select a Load Testing Tool:

Tools like JMeter, Locust, or Gatling are appropriate choices. For this strategy, we will consider using JMeter due to its versatility and ease of use.

3. Design the Load Test:

Endpoint1: This seems to be a static response, so the primary concern is how many requests the server can handle per second.

Endpoint2: The test should account for various input numbers, ensuring the calculation is consistent even under load.

Endpoint3: Focus on odd numbers mainly since even numbers throw an exception. However, including some even numbers is essential to understand how the system behaves under error conditions at scale.

4. Create Test Scenarios:

Baseline Test:

Use a small number of users, e.g., 10 users.

Measure the system's response times and error rates in a relatively low-load situation.

Ramp-Up Test:

Start with a small number of users and gradually increase over a predetermined period (e.g., 1 hour) until you reach your target max users.

This helps identify potential bottlenecks or breaking points.

Peak Load Test:

Simulate the maximum number of expected users all at once.

Maintain this load for a specified time (e.g., 30 minutes) to see if the system remains stable.

Endurance Test:

Run a continuous test for an extended period (e.g., 24 hours) with a high number of users to detect potential issues like memory leaks.

5. Test Execution:

Setup JMeter to simulate the desired scenarios.

Monitor server resources (CPU, memory, disk I/O, etc.) during the tests. Tools like Grafana integrated with Prometheus can help visualize performance metrics.

If possible, configure monitoring tools to get insights from the database, caching system, and other dependent services.

6. Analyze Results:

Review response times: Are they within acceptable limits?

Check error rates: Are there any unexpected spikes?

Analyze server resource utilization: Is there any unusual behavior like a memory leak?

Examine any failures or slowdowns to identify bottlenecks or weak points.

7. Optimization & Retesting:

Based on the test results, make necessary optimizations to the system.

After making changes, retest to validate the improvements.

8. Reporting:

Create a comprehensive report detailing test objectives, methods, results, and recommendations.

Conclusion:

A strategic approach to load testing ensures the service can handle real-world demands, offering an optimal user experience even during peak usage times. Given the nature of the provided endpoints, it's crucial not only to validate their ability to handle many requests but also to ensure calculations and error handling remain consistent under stress.