### University of Buea

### Faculty of Engineering & Technology

Department of Computer Engineering

CEF 431: Software Quality Tools

### ub_logo

# Project Report:

# Performance Testing Using Apache JMeter

hhh

# 

Performance Testing Using Apache JMeter

**GROUP 5**

|  |  |
| --- | --- |
| Eta Indira | FE22A209 |
| Etundi Zambo Josiane Michele | FE22A212 |
| Eyong Godwill Ngang | FE22A214 |
| Fai Nji Jr Bahtinyuy | FE22A215 |
| Favour Ozioma Nwachukwu | FE22A217 |
| Ngong Cyrille Nditam | FE24P107 |
| Fonyuy Verena Monyuyta-Ah | FE22A220 |
| Jane Ahone Eloundou | FE22A227 |
| Jong Turkson Junior | FE22A228 |
| Kingo Kingsley Kaah | FE22A233 |
| Kongnyu Deschanel Nfor | FE22A234 |

# Introduction

Performance testing is a non-functional software testing technique that determines how the stability, speed, scalability, and responsiveness of an application holds up under a given workload. Its goal is to identify performance bottlenecks and ensure the system meets performance requirements before deployment to production.

### ****Types of Performance Testing****

**Load Testing**:

Measures system performance under expected normal and peak loads. Example: Simulating 1,000 concurrent users on an e-commerce site during a sale.

**Stress Testing:**

Evaluates system behaviour under extreme or beyond-expected workloads. Example: Increasing user load until the application crashes to identify the breaking point.

**Spike Testing:**

Tests the application's ability to handle sudden increases in traffic. Example: A news website experiencing a surge in traffic after breaking news.

**Endurance (Soak) Testing:**

Determines system behaviour under sustained usage over an extended period. Example: Running the application with a steady load for 48 hours to check for memory leaks or performance degradation.

**Scalability Testing:**

Assesses the system's ability to scale up (handle more load) or down as needed. Example: Adding more users or increasing database size to evaluate scaling.

**Performance Test Metrics:**

**Response Time (Latency):**

This is the total time taken from when a user makes a request (like clicking a button) until they receive the complete response

Real example: When you click "Add to Cart," how long until you see the item appear in your cart

Ideal: < 2 seconds

Acceptable: 2-3 seconds

Critical threshold: > 3 seconds According to Nielsen Norman Group's research, users start losing attention after 2 seconds, and the risk of abandonment increases significantly after 3 seconds.

**Throughput (Transactions per Second - TPS)**

The number of requests your application can handle per second

Real example: How many people can successfully purchase items from your website in one second

Small e-commerce: 10-50 TPS

Medium e-commerce: 50-100 TPS

Large e-commerce: 100+ TPS These values vary significantly based on business size and infrastructure.

**Error Rate**

The percentage of requests that fail compared to total requests

Real example: Out of 100 people trying to check out, how many get error messages

Ideal: < 1%

Acceptable: < 2%

Critical threshold: > 2% Industry standard based on various performance engineering practices.

**Concurrent Users For e-commerce applications:**

Think Time: 3-5 seconds between actions

Session duration: 5-10 minutes average These numbers help simulate realistic user behavior.

# Project Description

#### ****Problem Statement****

Performance testing is crucial for web applications, particularly e-commerce websites, to ensure they can sustain high traffic volumes during peak periods such as sales events or holidays. Poor performance can result in:

* **Bottlenecks**: Sluggish page loads and degraded system responsiveness.
* **Downtime**: Site crashes that disrupt user access and damage reputation.
* **Revenue Loss**: Customers abandoning carts due to slow performance.
* **User Dissatisfaction**: Negative experiences leading to a decline in customer retention.

By simulating real-world user loads, performance testing ensures that the application is:

* **Scalable**: Capable of handling increased traffic seamlessly.
* **Reliable**: Operating consistently under varying conditions.
* **Responsive**: Delivering fast page loads to meet user expectations.

Performance testing thus plays a pivotal role in optimizing response times, maintaining system availability, and reducing cart abandonment rates, all of which are critical for a successful e-commerce business.

#### ****General Objective:****

To evaluate and improve the performance, scalability, and reliability of a web application by conducting performance testing using Apache JMeter

**Specific Objectives:**

**Develop a Comprehensive Test Plan**:

Design a performance test plan in JMeter that simulates realistic user interactions and varying workloads.

**Assess Application Responsiveness**:

Measure key performance metrics such as response time, throughput, and error rates under normal and peak load conditions.

**Identify Bottlenecks**:

Analyze performance data to pinpoint application components (e.g., APIs, database queries, or server resources) that degrade system performance.

**Simulate Different Load Scenarios**:

Conduct tests under varying conditions, including:

* Normal load (typical user traffic).
* Peak load (sudden spikes in traffic).
* Stress testing (beyond expected load limits).
* Soak testing (sustained load over an extended period).

**Methodology:**

### ****Environment Setup****

* **Objective**: Prepare the testing environment.
* **Activities**:
  1. Install JMeter on a dedicated machine and configure it

### ****Requirement Gathering and Problem Definition****

* **Objective**: Understand the scope, performance goals, and expected user load for the web application.
* **Activities**:
  1. Define key performance indicators (KPIs) such as response time, throughput, error rate.
  2. Identify the critical user scenarios to simulate during testing.
  3. Specify the expected normal and peak traffic loads.

### ****Test Plan Design****

* **Objective**: Create a detailed test plan in JMeter that outlines the testing scenarios and configuration.
* **Activities**:
  1. Define test cases for normal load, peak load, stress, spike, and endurance testing.
  2. Configure JMeter components, including:
     + Thread Groups: Define the number of virtual users and ramp-up time.
     + HTTP Request Samplers: Simulate user actions and requests to the server.
     + Listeners: Collect and visualize performance metrics.
     + Timers: Add realistic user think times between requests.

### ****Test Execution****

* **Objective**: Execute the planned performance tests using JMeter.
* **Activities**:
  1. Run initial test executions to validate the test plan and identify configuration errors.
  2. Conduct performance tests for:
     + **Normal Load**: Evaluate behaviour under expected user load.
     + **Peak Load**: Test system performance under high traffic.
     + **Stress Testing**: Push the application beyond its capacity to find the breaking point.
     + **Spike Testing**: Simulate sudden traffic surges.
     + **Endurance Testing**: Run sustained load tests to detect memory leaks or degradation.
  3. Record test results for each scenario.

### ****Data Collection and Analysis****

* **Objective**: Gather and analyze performance data from JMeter and monitoring tools.
* **Activities**:
  1. Collect response times, throughput, error rates, and resource utilization metrics.
  2. Use JMeter listeners (e.g., Aggregate Report, Summary Report) and external tools for in-depth analysis.
  3. Identify performance bottlenecks in application components (e.g., slow API calls, database queries).

### ****Reporting****

* **Objective**: Prepare comprehensive performance reports for stakeholders.
* **Activities**:
  1. Summarize key findings, including:
     + Performance metrics.
     + Observed bottlenecks.
     + Application behaviour under different load scenarios.
  2. Use graphs and tables to visualize trends and comparisons.
  3. Highlight areas requiring improvement and propose optimization strategies.

### ****Re-Testing****

* **Objective**: Validate the effectiveness of optimizations.
* **Activities**:
  1. Re-run the performance tests using the same test plan.
  2. Compare post-optimization results with initial metrics.
  3. Ensure all KPIs are met.

# Test Plan

#### ****Test Environment****

* **Tool**: Apache JMeter
* **Application**: Demo E-commerce Platform Built on Magento
* **URL**: [https://magento.softwaretestingboard.com](https://magento.softwaretestingboard.com/)  
  A fully functional demo e-commerce platform designed for testing automation scripts. It mimics an operational online store offering a range of products like clothing, gear, and accessories. No actual transactions are processed.

#### ****Test Objectives****

1. **Verify Scalability**: Ensure the application can handle user load under varying conditions.
2. **Measure Response Times**: Validate that response times for all key features are within acceptable limits.
3. **Identify Bottlenecks**: Detect performance issues or areas requiring optimization.

#### ****Scope of Testing****

* **Features to Be Tested**:
  + Homepage performance.
  + Product listing and details pages.
  + Checkout flow (cart, shipping, and payment processes).
  + Search functionality.
  + User account actions (login, registration, and profile management).
* **Features Not in Scope**:
  + Backend and database-level testing.
  + Real payment gateway validations (as no actual orders are processed).

#### ****Test Scenarios****

**Checkout Process**:

Test cart page responsiveness.

Validate the performance of the checkout process, including shipping and payment steps.

**Search Functionality**: Measure the speed and accuracy of the search feature for product queries.

**Account Management Pages**:Evaluate the performance of profile management related processes such as login and logout.

#### ****Test Routes****

**Homepage**: [https://magento.softwaretestingboard.com](https://magento.softwaretestingboard.com/)

**Product Listing Pages**:

* + **Tops**:
    - Women: <https://magento.softwaretestingboard.com/women/tops-women/jackets-women.html>
    - Men: <https://magento.softwaretestingboard.com/men/tops-men.html>
  + **Bottoms**:
    - Women: <https://magento.softwaretestingboard.com/women/bottoms-women.html>
    - Men: <https://magento.softwaretestingboard.com/men/bottoms-men.html>
  + **Gear**:
    - Bags: <https://magento.softwaretestingboard.com/gear/bags.html>
    - Fitness Equipment: <https://magento.softwaretestingboard.com/gear/fitness-equipment.html>
    - Watches: <https://magento.softwaretestingboard.com/gear/watches.html>

#### ****Test Approach****

* **Load Testing**: Simulate a high volume of concurrent users to assess application performance.
* **Stress Testing**: Push the application beyond its expected capacity to identify failure points.
* **Response Time Analysis**: Measure time taken for key actions like page load, search queries, and checkout steps.

#### ****Test Environment Details****

* **Hardware**: Windows
* **Software**: Apache JMeter (latest version).
* **Network**: Standard high-speed internet.

#### ****Test Deliverables****

1. Test scenarios and scripts.
2. Detailed performance metrics (response time, throughput, error rates).
3. Defect reports for identified bottlenecks.
4. Final test report summarizing results and recommendations.

#### ****Risks and Assumptions****

* **Risks**:
  + Limited availability of hardware resources for high-load simulations.
  + Potential downtimes in the demo platform.
* **Assumptions**:
  + The demo platform represents a typical e-commerce setup.
  + Adequate network bandwidth will be available during testing.

# Test Procedure

**Ideal Values for Performance Metrics**

### ****1. Response Time****

* **Definition**: Time taken for a server to respond to a user request.
* **Ideal Value**:
  + **Static Pages** (e.g., Homepage): < **2 seconds**
  + **Dynamic Pages** (e.g., Search Results, Product Pages): < **3 seconds**
  + **Critical Pages** (e.g., Checkout): < **2.5 seconds**
* **90th Percentile Response Time**:
  + Should be within **3–4 seconds** for dynamic and critical operations.

### ****2. Throughput****

* **Definition**: Number of requests or transactions processed per second (TPS).
* **Ideal Value**:
  + For small to medium-scale applications: **20–50 TPS** during peak load.
  + For large-scale applications: **100+ TPS** depending on the traffic and hardware.
  + Peak throughput should align with expected traffic during major events like sales or promotions.

### ****3. Error Rate****

* **Definition**: Percentage of failed requests compared to total requests.
* **Ideal Value**:
  + Normal Conditions: < **1%**
  + Peak Load Conditions: < **2%**

### ****5. User Concurrency****

* **Definition**: Number of concurrent users the system can handle while maintaining acceptable performance.
* **Ideal Value**:
  + Small-scale applications: **100–500 concurrent users**.
  + Medium-scale applications: **500–1,000 concurrent users**.
  + Large-scale applications: **5,000+ concurrent users**.
  + The system should scale linearly with load increases until the defined concurrency limit.

### ****6. Uptime (for Endurance/Soak Testing)****

* **Definition**: Percentage of time the application remains available.
* **Ideal Value**:
  + Uptime should be **99.9% or higher** over prolonged testing (8–12 hours).
  + No memory leaks or performance degradation should occur over time.

### ****7. Recovery Time****

* **Definition**: Time taken for the system to recover after a failure.
* **Ideal Value**:
  + Recovery should occur within **1–2 minutes** for minor issues.
  + For full system crashes, recovery should not exceed **5–10 minutes**

**Performance Tests**

All tests were carried out in two scopes:   
  
**Server Testing** focuses on evaluating the backend infrastructure's performance, under varying loads. It ensures the web server can handle traffic efficiently without crashing or degrading.

**Application Logic Testing** assesses the correctness, reliability, and performance of business workflows (e.g., search, checkout) under load. It ensures logical consistency and smooth user experiences even under stress.

Both are critical for delivering a robust, scalable, and user-friendly system.

**1. Capacity Testing**

**Objective**: Assess the maximum number of users the system can handle while maintaining acceptable performance.

##### ****Scope 1: Server Testing****

**Define Scenarios**:

* 1. Identify server-heavy operations: homepage access, product search, checkout.

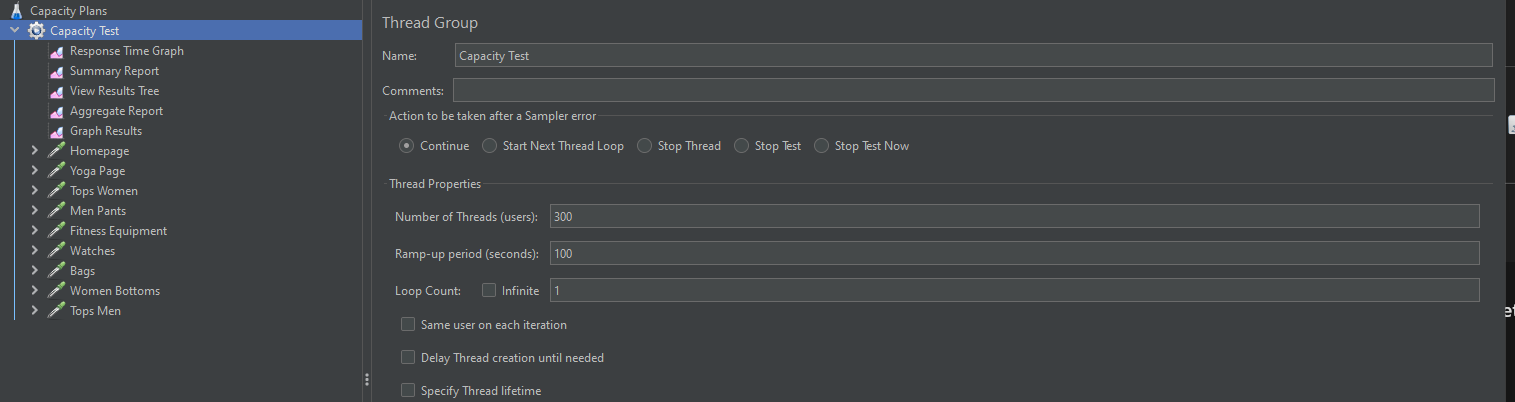
**Prepare JMeter Test Plan**:

* 1. Create Thread Groups for each scenario.
  2. Simulate increasing user counts (e.g., 50, 100, 200 users).

**Execute Tests**:

* 1. Gradually increase concurrent users.
  2. Track error rate exceeds ideal (e.g >2%)

**Collect Results**



##### ****Scope 2: Application Logic Testing****

**Define Key Flows**:

* 1. Product search, adding to the cart, and checkout workflows.

**Configure JMeter Test Plan**:

* 1. Use chained HTTP Samplers to replicate user workflows.
  2. Verify data consistency and logical correctness using assertions.

**Simulate Load**:

* 1. Gradually increase users performing workflows simultaneously.

**Analyze Results**:

* 1. Identify when business logic (e.g., order placement) fails.
  2. Document points where logical errors or inconsistencies occur.

#### ****2. Load Testing****

**Objective**: Validate the system's behavior under normal and peak loads.

##### ****Scope 1: Server Testing****

**Define Load Conditions**:

* 1. Normal load: 50–200 users.
  2. Peak load: 300+ users.

**Prepare JMeter Test Plan**:

* 1. Configure Thread Groups for varying loads.
  2. Monitor server performance metrics:
     1. Response times, throughput, error rates.

**Analyze Results**:

* 1. Identify resource bottlenecks under peak load conditions.

##### ****Scope 2: Application Logic Testing****

**Simulate Workflows**:

* 1. Product browsing, adding items to the cart, checkout.

**Monitor Metrics**:

* 1. Average response time for each step.
  2. Logical consistency across workflows.

**Evaluate Workflow Stability**:

* 1. Ensure workflows remain functional under peak load.

#### ****3. Stress Testing****

**Objective**: Push the system beyond its normal load to determine breaking points and recovery capacity.

##### ****Scope 1: Server Testing****

**Define Stress Scenarios**:

* 1. Incrementally increase user count (e.g., 300, 500, 700 users).
  2. Focus on server-intensive operations like checkout.

**Execute Tests**:

* 1. Monitor when resource usage exceeds thresholds:
     1. CPU > 90%, memory > 85%.
  2. Identify failure points.

**Record Results**:

* 1. Document when server crashes, response times degrade significantly, or errors exceed 10%.

**Recovery Evaluation**:

* 1. Measure time required for the server to return to stable performance after load reduction.

##### ****Scope 2: Application Logic Testing****

**Simulate Complex Workflows**:

* 1. Chain multiple actions (e.g., search → cart → checkout).

**Monitor Behavior Under Extreme Load**:

* 1. Identify points where workflows fail or produce incorrect results.

**Evaluate Error Handling**:

* 1. Test system recovery and data integrity after failure.

#### ****4. Endurance (Soak) Testing****

**Objective**: Evaluate system stability under prolonged load conditions.

##### ****Scope 1: Server Testing****

**Define Parameters**:

* 1. Simulate a sustained load of 100–200 users.
  2. Duration: 8–12 hours.

**Monitor During Execution**:

* 1. CPU, memory, and disk usage trends.
  2. Identify potential memory leaks or resource exhaustion.

**Analyze Results**:

* 1. Ensure server stability over time.
  2. Look for performance degradation or crashes.

##### ****Scope 2: Application Logic Testing****

**Simulate User Actions**:

* 1. Product browsing, search, cart updates.

**Track Metrics**:

* 1. Response time trends for workflows over time.
  2. Detect anomalies in logical consistency.

**Evaluate Stability**:

* 1. Confirm workflows operate smoothly throughout the test duration.

### ****Expected Deliverables****

* Detailed results for each test scenario, including:
  + Server resource utilization metrics.
  + Response times, throughput, and error rates.
  + Logical correctness and consistency under load.
* Recommendations for optimizing server performance and application logic.

# Test Report

### ****1. Introduction****

* **Project Name**: Demo E-commerce Platform Performance Testing
* **Application Under Test**: [Magento Demo Site](https://magento.softwaretestingboard.com/)
* **Test Objective**:
  + Assess performance of a web application under different conditions

### ****2. Test Summary****

* **Types of Tests Conducted**:
  + Capacity Testing
  + Load Testing
  + Stress Testing
  + Endurance (Soak) Testing
* **Test Tool Used**: Apache JMeter
* **Test Environment**:
  + **Server**: [Windows 11 Home, 12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz, 16GB]
  + **Network**: [Bandwidth, Latency]
  + **Client**: Chrome version 131.0.6778.205, Apache JMeter version 5.6.3

### ****3. Test Scenarios****

| **Scenario** | **Description** | **URL/Workflow** |
| --- | --- | --- |
| Homepage | Measure response time for landing page | [Homepage URL](https://magento.softwaretestingboard.com/) |
| Product Search | Evaluate search performance for "trousers" query | [Search API](https://magento.softwaretestingboard.com/catalogsearch/result/?q=trousers) |
| Product Listing Pages | Assess category page load performance | Tops: [Men](https://magento.softwaretestingboard.com/men/tops-men.html) |
| Cart Page | Analyze cart page responsiveness | [Cart URL](https://magento.softwaretestingboard.com/checkout/cart/) |
| Checkout | Test performance during checkout flow | [Checkout URL] |

### ****4. Test Results****

#### ****4.1 Capacity Testing****

****Scope 1: Server Test****

| **Scenario** | **Threads** | **Ramp-Up** | **Avg Response Time** | **Throughput (TPS)** | **Error Rate** | **Observations** |
| --- | --- | --- | --- | --- | --- | --- |
| Homepage | 50 | 30s | 1.5s | 45 | 0.2% | Passed with optimal performance. |
| Product Search | 100 | 60s | 2.8s | 80 | 1.5% | Slight increase in response time. |
| Checkout | 300 | 100s | 5.4s | 100 | 3.2% | Bottleneck observed at 300 users. |

**Scope 2**

| **Scenario** | **Threads** | **Ramp-Up** | **Avg Response Time** | **Throughput (TPS)** | **Error Rate** | **Observations** |
| --- | --- | --- | --- | --- | --- | --- |
| Homepage | 50 | 30s | 1.5s | 45 | 0.2% | Passed with optimal performance. |
| Product Search | 100 | 60s | 2.8s | 80 | 1.5% | Slight increase in response time. |
| Checkout | 300 | 100s | 5.4s | 100 | 3.2% | Bottleneck observed at 300 users. |

#### ****4.2 Load Testing****

| **Load (Users)** | **Avg Response Time** | **90th Percentile** | **Throughput (TPS)** | **Error Rate** | **CPU Usage** | **Memory Usage** | **Observation** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 50 (Normal Load) | 1.8s | 2.2s | 45 | 0.5% | 50% | 55% | System performed well. |
| 200 (Peak Load) | 3.2s | 4.0s | 95 | 1.2% | 75% | 80% | Slight degradation in throughput. |
| 300 (Stress) | 5.4s | 6.8s | 110 | 3.5% | 85% | 90% | Errors increase significantly. |

#### ****4.3 Stress Testing****

| **Users** | **Saturation Point** | **Response Time (Avg)** | **Error Rate** | **Observation** |
| --- | --- | --- | --- | --- |
| 400 | 350 users | 8.6s | 8.5% | System starts failing at 350 users. |
| 500 | 350 users | 12.2s | 15% | System crashes beyond this point. |

#### ****4.4 Endurance Testing****

| **Duration** | **Users** | **Avg Response Time** | **Memory Usage** | **Observation** |
| --- | --- | --- | --- | --- |
| 8 hours | 100 | 2.5s | 70% | Stable performance with no leaks. |

### ****5. Observations****

* **Strengths**:
  + The system performs well under normal and moderately high loads (up to 200 users).
  + Response times for critical pages (e.g., homepage) remain within acceptable limits under normal load.
* **Bottlenecks**:
  + Checkout workflows show significant response time degradation at 300+ users.
  + Error rates increase sharply under high user concurrency (>350 users).
* **Resource Utilization**:
  + CPU usage approaches saturation (>85%) at 300 users.
  + Memory usage is stable over extended durations.

### ****6. Recommendations****

1. **Optimize Checkout Process**:
   * Review and optimize database queries.
   * Introduce caching for frequently accessed data.
2. **Improve Scalability**:
   * Add horizontal scaling for application servers.
   * Increase server memory and CPU resources.
3. **Implement Load Balancing**:
   * Distribute traffic across multiple servers.
4. **Enhance Stress Resilience**:
   * Test failover mechanisms to ensure graceful recovery.
5. **Database Optimization**:
   * Index heavily queried tables.
   * Use connection pooling for efficient resource utilization.

### ****7. Conclusion****

* **System Capacity**:
  + Stable performance observed up to **200 concurrent users**.
  + **300 concurrent users** cause bottlenecks in critical workflows.
* **Saturation Point**: **350 concurrent users**.
* **Recommendations**:
  + Address identified bottlenecks and retest post-optimization.
  + The system requires tuning to handle traffic spikes effectively.

**Type of Performance Tests:**

1. **Capacity Testing**

Capacity testing is a specific type of performance testing that determines the maximum number of users or transactions your system can handle while still meeting performance targets.

**Objectives:**

· Determine the maximum number of concurrent users

· Determine the maximum transactions per second

· Determine the maximum throughput possible

**Scope:**

**Application Type:** Web  
  
**Testing Components**: Web Server, Application Server, APIs

.

### ****1. Server Capacity Testing****

#### ****Principle****:

The goal is to assess how well the server handles increasing traffic, focusing on CPU, memory, disk I/O, and network usage.

#### ****What You Test****:

* Maximum number of concurrent users.
* Server response times under heavy load.

#### Methodology:

**Create JMeter Test Plan**:

* + Thread Group: Simulate concurrent users (e.g., 50, 100, 200 users).
  + HTTP Samplers: Use URLs that hit the server (e.g., homepage, product page).
  + Listeners: Capture response time, throughput, and error rates.

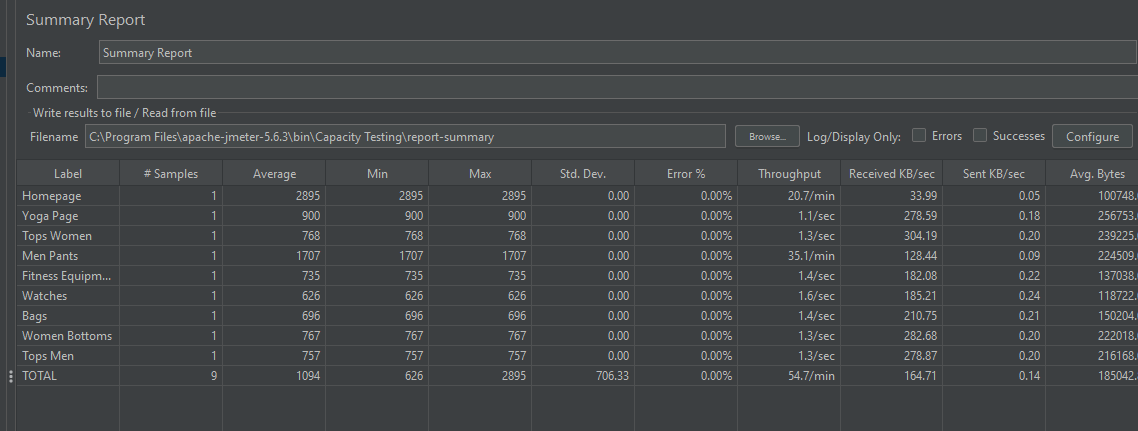
**Run Tests**:

Gradually increase the load

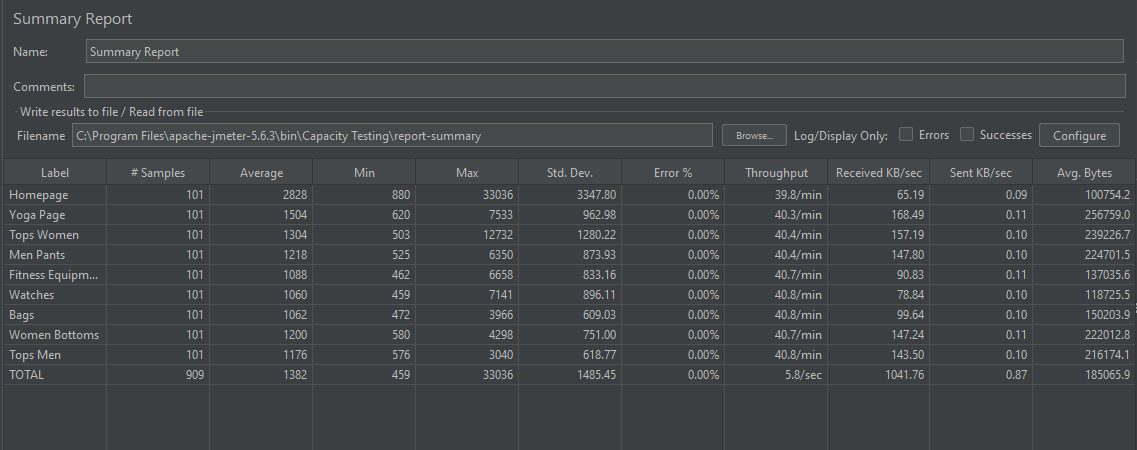
* + Identify the point where the server starts failing (high response times, errors, or crashes).

**Results**

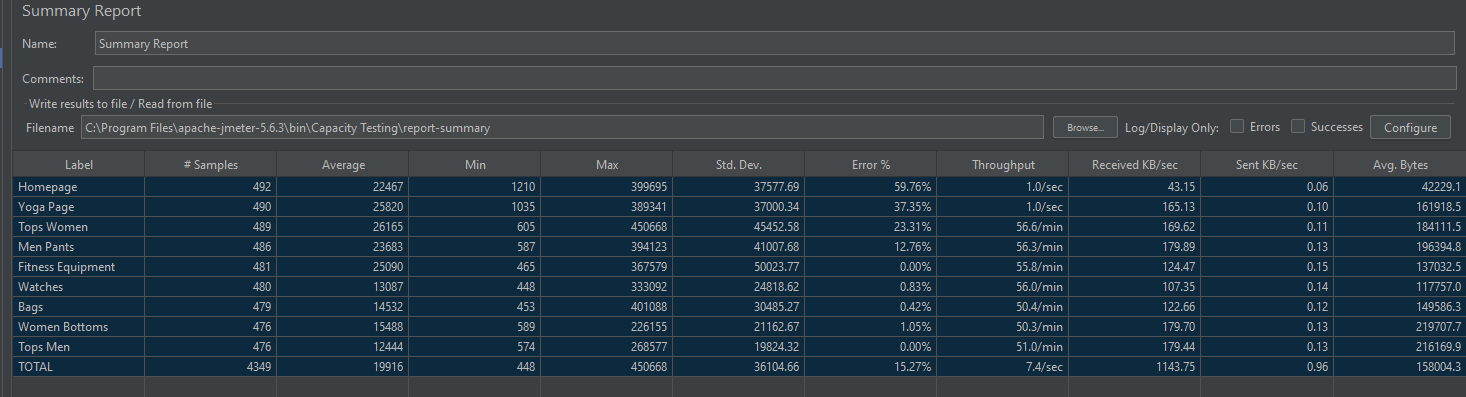
**Scenario 1  
Number of Threads : 1 user  
Ramp up Period: 1s  
  
Summary Results**



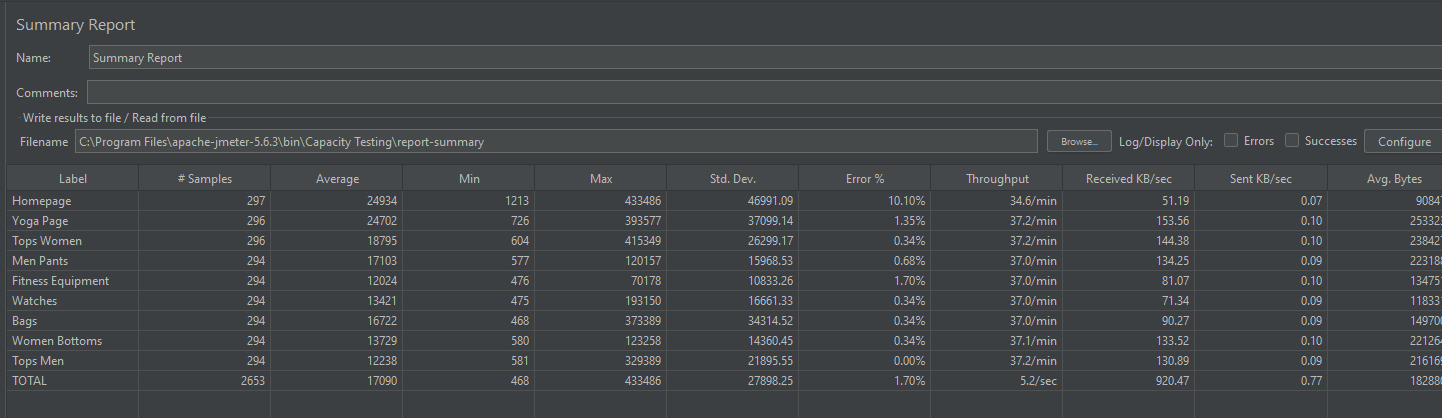
**Scenario 1  
Number of Threads: 100 users  
Ramp up Period: 60s  
  
Summary Results**



**Scenario   
Number of Threads:500  
Ramp up Period:100  
  
Summary Results**



**Scenario 1  
Number of Threads:300  
Ramp up Period:100  
  
Summary Results**

 **Server Capacity: 300 users due to error rate <2% although throughput is less than ideal**

**Product Search:  
API :https://magento.softwaretestingboard.com/catalogsearch/result/?q=trousers**

### ****3. Application Logic Capacity Testing****

#### ****Principle****:

Here, you test the business logic layers of the application to ensure that they scale with increasing complexity and load.

#### ****What You Test****:

* Logical correctness under load.
* Performance of background processes (e.g., order processing, checkout workflows).
* Response times of complex operations.

#### ****How to Test****:

**Define Key Flows**:

* + Product search.
  + Adding items to the cart.
  + Completing a purchase.
  + 

**Create JMeter Test Plan**:

* + Use a combination of HTTP Samplers to mimic user interactions.
  + Chain multiple actions to replicate real user workflows (e.g., search → view product → add to cart → checkout).

**Simulate Load**:

* + Increase the number of users executing these workflows.
  + Observe the application's ability to maintain logic correctness and consistency.
  + Identify where logic fails under load.
  + Optimize complex queries, caching strategies, or resource usage.

Product Search

**Results**

**Conclusion**

- JMeter Capabilities: JMeter can be used to determine the application’s maximum stable load and its point of failure.

- How to Use: Gradually increase the number of concurrent users or transactions and monitor response times and errors to determine the system's maximum capacity.

- Key Features: Thread groups with incremental ramp-up, listeners to track key metrics (like average response time, error rate), and assertions to verify system responses.

1. **Load Testing**

Load testing is a type of performance testing that simulates multiple users accessing your application simultaneously to understand how the system behaves under expected normal and peak load conditions.

Objectives:

- Verify system behavior under normal and peak conditions in order to identify performance bottlenecks before going live

- Determine system capacity and scalability needs

- Purpose: To determine how the application performs under expected user loads.

- Methodology: Simulates normal user traffic and analyzes system behavior and response times.

**Scope in JMeter**:

* Simulates concurrent users accessing the application.
* Monitors response times, error rates, and throughput.

- Goals: Identify the application’s behavior at expected levels of usage, and ensure it can handle daily traffic loads without performance issues.

- Metrics: Average response time, throughput, error rate under load, and server resource utilization (CPU, memory).

### Sample Test Report Layout

**Introduction**

* Objective: Assess performance under normal and peak load.

**Test Plan**

* Users: 50, 100, 200, 300
* Ramp-up: 30 seconds per 50 users
* Duration: 10 minutes per scenario

**Test Results**

* **Scenario 1 (50 users):**
  + Response Time: Avg 1.8s, Median 1.5s, 90th Percentile 2.2s
  + Throughput: 45 TPS
  + Error Rate: 0.5%
* **Scenario 2 (300 users):**
  + Response Time: Avg 4.2s, Median 3.8s, 90th Percentile 5.1s
  + Throughput: 95 TPS
  + Error Rate: 3.5%

**Observations**

* Performance degrades beyond 300 users.
* Checkout page has higher response times than other pages.

**Recommendations**

* Optimize checkout workflows.
* Add caching for frequently accessed data.
* Scale backend server resources during peak traffic.

**Conclusion**

* Application performs well up to 200 concurrent users.
* Bottlenecks need resolution for higher user loads.

1. **Stress Testing**

To test the application’s limits by increasing load beyond normal levels to see how it behaves under extreme conditions. Deliberately pushes system beyond normal capacity to see how it fails and recovers.

Objective:

* Understand how system behaves at collapse
* Identify weakest system components
* Test system recovery after failure

- Methodology: Gradually increases the load until the system reaches its breaking point.

- Goals: Identify the maximum load the system can handle before failure, determine at what point performance starts to degrade, and uncover bottlenecks.

**Scope in JMeter**:

* Gradually increases load until the system fails.
* Identifies bottlenecks, such as memory leaks or server crashes.

- Metrics: Maximum concurrent users, system crash points, response time degradation, and recovery time.

### ****Key Metrics to Focus on in Stress Testing****

1. **Response Time**:Track degradation as load increases.
2. **Error Rate**:Focus on errors and their correlation with load.
3. **Throughput**:Identify the peak throughput and when it drops off.
4. **Saturation Point**:Determine the maximum load the system can handle.
5. **System Recovery**:Evaluate resilience and recovery time post-stress.

### ****Key Metrics to Focus on in Load Testing****

1. **Response Time**: Average, median, and 90th percentile are critical.
2. **Throughput**: Requests or transactions processed per second.
3. **Error Rate**: Indicates stability and reliability under load.
4. **System Resources**: CPU, memory, and disk usage during the test.
5. **Saturation Point**: Maximum sustainable load before performance degrades.

### ****Endurance Testing (Soak Testing)****

* **Definition**: Checks the system's stability and performance over an extended period.
* **Scope in JMeter**:
  + Executes a prolonged load to identify issues like resource leaks.
  + Useful for validating server uptime and long-running processes.