|  |
| --- |
|  |
| Software Quality Engineering  Assignment 4 |
| |  |  |  | | --- | --- | --- | | NADEEM AHMAD,HAMDAN AHMAD | 22F-3679,22F3678 |  | |

**TASK:1**

**Boundary Value Analysis:**

* **Lower Boundary**: GET /api/users?page=1
* **Upper Boundary**: GET /api/users?page=3
* **Invalid Boundary**: Negative values GET /api/users?page=-1

**Equivalence Partitioning:**

* **Valid Partition**: Pages 1 to 3.
* **Invalid Partition**: Pages beyond available (e.g., page=999) or negative page numbers.

**Error Guessing:**

* Send an invalid data type for the page number (e.g., GET /api/users?page=abc).
* Send an empty page value (e.g., GET /api/users?page=).

**Clarifying "4 Endpoints"**

* We have **four distinct endpoint categories**:
  1. **Users Endpoint** (Manipulate user data)
  2. **Resources Endpoint** (Manipulate resource data)
  3. **Register/Login Endpoint** (For authentication)
  4. **Delayed Response Endpoint** (To test network latency scenarios)

**TASK2:**

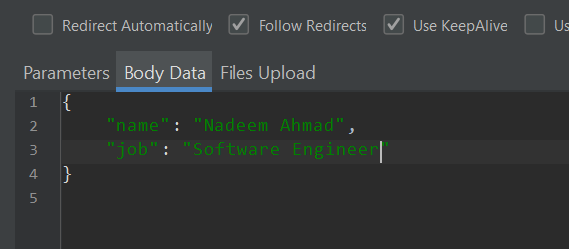
**Scenario Overview**

Let's set up **2 load test scenarios** using JMeter based on your existing API testing:

1. **Scenario 1** - **Load Test for GET Users List**: This will involve sending multiple GET requests to retrieve the users' list and check server performance under load.
2. **Scenario 2** - **Load Test for CRUD Operations (POST, PUT, DELETE)**: This scenario will involve performing multiple POST, PUT, and DELETE operations to simulate load on the server for creating, updating, and deleting a user.

* **Configure the Thread Group**:
  + **Number of Threads (users)**: This is the number of virtual users. Set to 10 for a moderate load.
  + **Ramp-Up Period (in seconds)**: The time taken to start all users. Set to 5.
  + **Loop Count**: Set to 10 for a fixed number of iterations.

This configuration will simulate 10 users performing each scenario in 10 loops, resulting in a total of 100 requests for each endpoint.



**Video of SETUP AND RUNNING PROGRAM**

**Get method setup**

[**F:\5 semester\sqe\videos\get method.mp4**](file:///F:\5%20semester\sqe\videos\get%20method.mp4)

**Post method setup**

[**F:\5 semester\sqe\videos\post.mp4**](file:///F:\5%20semester\sqe\videos\post.mp4)

**Put method setup**

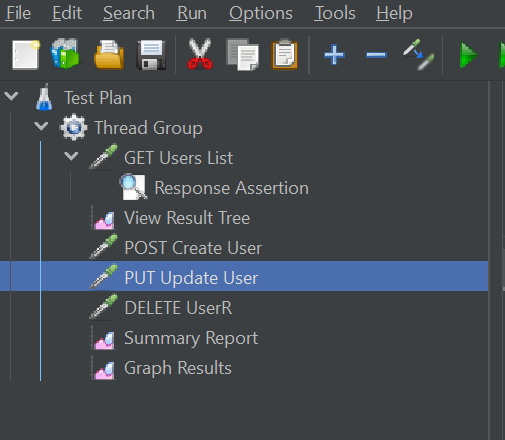
[**F:\5 semester\sqe\videos\put.mp4**](file:///F:\5%20semester\sqe\videos\put.mp4)

**Delete method setup**

[**F:\5 semester\sqe\videos\delete method.mp4**](file:///F:\5%20semester\sqe\videos\delete%20method.mp4)

**Final Running**

[**F:\5 semester\sqe\videos\running jmeter.mp4**](file:///F:\5%20semester\sqe\videos\running%20jmeter.mp4)



**TASK3:**

**1---7 Performance**

**Remaining Security**

|  |  |
| --- | --- |
| **SR NO** | **NON-FUNCTIONAL REQUIREMENTS** |
|  |  |
| **NFR 1** | Ensure that the API responds to requests within 200ms for 95% of all requests under normal load conditions. Aim for <100ms for critical endpoints. |
| **NFR 2** | The API should handle at least 1,000 concurrent users with a 0.1% error rate. The platform must support high traffic with no degradation in user experience. |
| **NFR 3** | The internal processing latency should be under 50ms to maintain fast data processing for the users. Minimize delays within server-side operations |
| **NFR 4** | The API should maintain an error rate of less than 0.1% during both normal and peak load. This minimizes failed transactions or broken sessions for users. |
| **NFR 5** | The system must achieve a 99.9% uptime to ensure constant availability for users, including maintenance windows. Downtime should be limited to scheduled periods. |
| **NFR 6** | Utilize data compression techniques such as gzip to ensure the payload size is <1MB whenever feasible. This reduces bandwidth and speeds up data transfers. |
| **NFR 7** | Implement server-side and client-side caching to ensure a cache hit rate between 60-80% for frequently accessed data, improving response times for users. |
| **NFR 8** | Implement OAuth 2.0 as the authentication mechanism to securely validate user identity. Require strong passwords and multi-factor authentication (MFA). |
| **NFR 9** | Use Role-Based Access Control (RBAC) to define and manage user permissions. Only authorized users should have access to sensitive data and endpoints. |
| **NFR 10** | Enforce HTTPS using TLS 1.2 or higher to secure communication between the client and the server, ensuring that data is not intercepted or tampered with. |
| **NFR 11** | Validate and sanitize all incoming data to prevent injection attacks. Use whitelisting for expected inputs and limit data to only the required fields. |
| **NFR 12** | Implement rate limiting to restrict the number of API calls per user (e.g., 1000 requests/hour) to prevent abuse or Distributed Denial of Service (DDoS) attacks. |
| **NFR 13** | Use short-lived JWT (JSON Web Tokens) for managing user sessions. Ensure proper token expiration and renewal to prevent session hijacking. |
| **NFR 14** | Avoid detailed error messages that reveal system information. Use generic error messages like "500 - Internal Server Error" to prevent leaking system details. |

**Test-Plan**

**1. High-Level Expectations**

* **Goal**: Ensure the e-commerce platform is secure, handles sensitive data correctly, and supports up to 1,000 concurrent users without performance degradation.
* **Outcome**: All functional and non-functional requirements will be verified, including data security, performance, and system stability under load.
* **Scope**: The test will cover both frontend and backend components, focusing on core functionalities like user registration, login, product browsing, purchasing, and payment processing.

**2. People, Places, and Things**

* **Stakeholders**:
  + Project Manager
  + Development Team Lead
  + Test Manager
  + Security Specialist
  + DevOps Engineer
  + Product Owner
* **Contact Details**: Collect and document the email and phone contacts for each stakeholder.
* **Location**: Office space for developers and testers, remote access for off-site testing, secure lab for sensitive data testing.

**3. Definitions**

* **Quality Goals**: Ensure 99.9% uptime, zero security breaches, and an error-free user experience.
* **Reliability Goals**: The system should maintain stable performance with no more than 2% performance degradation under peak load.

**4. Inter-Group Responsibilities**

* **Developers**: Build features and fix bugs reported by testers.
* **Testers**: Create, execute, and maintain test cases. Provide reports on bugs and overall quality.
* **DevOps**: Maintain test environments, CI pipelines, and infrastructure.
* **Security Team**: Conduct penetration testing and provide security audits.

**5. What Will and Won’t Be Tested**

* **Tested**: User authentication, product catalog, cart functionality, checkout, payment processing, search capabilities, and API performance.
* **Not Tested**: Standard libraries, third-party payment gateways already certified, and static content such as terms and conditions.

**Test Phases**

**1. Unit Testing**

* Focus: Individual components, methods, and classes.
* Tools: JUnit, Mockito.

**2. Integration Testing**

* Focus: Interaction between backend APIs, frontend, and database.
* Tools: Postman, Rest Assured.

**3. System Testing**

* Focus: Complete system behavior, including front-end and back-end integration.
* Tools: Selenium WebDriver.

**4. Performance Testing**

* Focus: Load, stress, and scalability of APIs.
* Tools: Apache JMeter, K6.

**5. Security Testing**

* Focus: OWASP vulnerabilities such as SQL injection, XSS, CSRF.
* Tools: OWASP ZAP, Burp Suite, Postman.

**6. Acceptance Testing**

* Focus: Ensure system meets requirements from the user’s perspective.
* Tools: Manual testing, Cucumber for BDD.

**Test Strategy**

**1. Functional Testing**

* **Objective**: Verify that all features perform as expected.
* **Tools**: Postman (API testing), Selenium (UI testing).

**2. Load Testing**

* **Objective**: Simulate 1,000 concurrent users interacting with the platform.
* **Tools**: Apache JMeter.

**3. Security Testing**

* **Objective**: Ensure that sensitive data is secure, no vulnerabilities exist.
* **Tools**: OWASP ZAP.

**4. CI/CD Integration**

* **Tools**: Jenkins for automation pipelines, Git for version control.
* **Process**: Automated tests triggered on code commit, generating reports using tools like Allure.

**Resource Requirements**

* **People**:
  + 2 Backend Testers, 2 Frontend Testers, 1 Test Automation Engineer, 1 Security Tester.
* **Equipment**: High-performance machines, test servers, secure database.
* **Software**: Jenkins, Git, JUnit, Postman, Apache JMeter, OWASP ZAP.
* **Outsource**: Potential outsourcing for performance testing if internal capacity is insufficient.

**Tester Assignments**

* **Functional Tests**: Frontend Testers.
* **Security Tests**: Security Tester.
* **Performance Tests**: Test Automation Engineer.
* **Integration Tests**: Backend Testers.

**Test Schedule**

* **Unit Testing**: Week 1-2
* **Integration Testing**: Week 3-4
* **System Testing**: Week 5-6
* **Performance Testing**: Week 7-8
* **Security Testing**: Week 9
* **User Acceptance Testing**: Week 10

**Test Cases**

* Use **Test Management Tool** like TestRail or Jira for managing test cases.
* Focus on critical paths, boundary conditions, and high-risk areas.
* Test case management will involve mapping each requirement to specific tests.

**Bug Reporting**

* Use Jira or GitHub Issues.
* Bug severity levels: Critical, Major, Minor.
* Reports should include: Steps to reproduce, expected vs. actual behavior, screenshots/logs, environment details.

**Metrics and Statistics**

* **KPIs**:
  + Test coverage.
  + Number of bugs found per phase.
  + Bug fix rate.
  + Load capacity and system stability statistics.
  + Response times under peak load.
* **Monitoring**: Use Jenkins to monitor test automation results, generate daily reports, and track progress.

**Risks and Issues**

* **Deadlines**: Unrealistic deadlines could result in incomplete testing.
* **Skill Levels**: Insufficient expertise in security testing could miss vulnerabilities.
* **Test Environment**: Delays in setting up test environments or access to staging servers.
* **Resource Shortages**: Lack of high-capacity servers for performance testing.
* **Staff Turnover**: Key members leaving mid-project.

**Performance Requirements (Web API Benchmarks)**

1. **Response Time**: Less than 200ms for 95% of requests.
2. **Throughput**: Ability to handle at least 1000 concurrent users.
3. **Latency**: Below 50ms internal latency.
4. **Error Rate**: Less than 0.1% errors under normal load.

**Security Requirements (OWASP REST API Security)**

1. **Authentication**: Use OAuth 2.0 for user authentication.
2. **Transport Layer Security**: Enforce HTTPS with strong encryption (TLS 1.2 or higher).
3. **Data Validation**: Sanitize inputs to prevent injection attacks.
4. **Session Management**: Implement secure, short-lived session tokens.
5. **Error Handling**: Ensure error responses don’t leak sensitive information.

**Study Resources & Tools**

* **OWASP REST API Security**:
  + OWASP API Security Top 10
  + OWASP REST Security Cheat Sheet
* **Performance**:
  + API Performance Testing Guide
  + [Load Testing with JMeter](https://jmeter.apache.org/)