The BLOG

Performance Testing Strategy

1. Introduction

The purpose of this document is to specifically describe how the performance requirements for TheBlog application will be tested and verified in test (Jmeter) environment (System Under Test, SUT). The document will outline the scenarios, tests, parameters and data used in evaluating the capacity of the included features. The scope of tests described in current document is to verify how SUT behaves under load stress. All deliverables should be used for internal (in-house) usage only and should not be presented to public.

This document describes the strategy of performance testing for the TheBlog project. It consists of outlines for the following items:

* Scope of testing, test descriptions
* Non-functional requirements (NFR) related to performance
* Pass/fail criteria
* Approach
* Main scenarios
* Test cases
* Test data
* Requirements for test environment

1. Items to be tested

Mainly, TheBlog application will be tested via UI, as server side as client side.

|  |  |
| --- | --- |
| **#** | **Modules name** |
|  | Main script |
|  | Anonymous script |
|  | Open Post script |
|  | Admin script |
|  | Editor script |

1. Items not to be tested

Functionality of the whole application

1. Approach
   1. Test types assumed for conducting

The following activities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Test type** |  | **Period** | **Notes** |
|  | Smoke test | Mandatory | Regular |  |
|  | Capacity | Mandatory | After significant changes |  |
|  | Load test | Mandatory | Regular |  |
|  | Stress | Optional | Rare |  |
|  | Scalability | Mandatory | Once+ |  |
|  | Volume | Mandatory | Rare (once+) |  |
|  | Durable | Mandatory | Rare |  |
|  | Configuration | Mandatory | Rare (once+) |  |

* + 1. Smoke testing

Should be performed every time when functionality of the application and the script need to be checked. Also, if needed can be used as warming up test before main testing step.

* + 1. Capacity testing

Should be performed to find the number of virtual users which the application support in stable state. The test can be performed as one of first main tests, and should be performed after significant changes in the application or its configuration.

* + 1. Load test

Load test is a kind of the most regular tests to check benchmark of the application and its components. Usually, is running after finding

* + 1. Stress testing

Stress testing supposed to run occasionally to check application’s stability under high load. Can be performed close to after code complete or by special request.

* + 1. Scalability testing

Can be performed once or more in order to get multiplier(s) for different number of front-end/other servers.

* + 1. Volume testing

Is to run with small/planned/huge amount of data with regular load to get indicators on application’s responsiveness/metrics change. Should be performed at least once or by special request.

* + 1. Durable(Stability) testing

Supposed long time running the test with the load lower than average. Should be performed occasionally after significant code changes or by special request to make sure the application’s responsiveness and key performance indicators do not change significantly after long time running, and to check on memory leak as well.

* 1. Level of testing

The following levels of performance testing are supposed:

|  |  |  |
| --- | --- | --- |
| **#** | **Level** | **Description** |
| 1 | Purely mocked version | Used to test only application’s tier with no engaged services: basic client-side testing, smoke testing, checking test scripts etc. |
| 2 | Partially mocked version | Used to test particular set(s) of application’s components with test data to get their benchmark with no additional load for mocked parts. |
| 3 | Full set of test data | Used for end-to-end testing of the system |

1. Non-Functional requirements:

* **Speed** (The landing page supporting 5,000 users per hour must provide 4 second or less response time in a Chrome desktop browser, including the rendering of text and images and over an LTE connection.)
* **Scalability** (The system must be scalable enough to support 5,000 visits at the same time while maintaining optimal performance.)
* **Portability** (A program running on Windows 10 must be able to run on Windows 11 without any change in its behavior and performance.)
* **Security** (Account creation: Systems may require users to create accounts to access applications that store information and display profiles. A security system typically grants access to accounts when users enter the correct username and password; Password generation: An application may not grant access until the user creates a strong password (a strong password might contain a certain number of characters, a capital letter, and a special character.)
* **Usability** (users can easily navigate in application interface)
* **Reliability** (The system must perform without failure in 95 percent of use cases during a month.)
* **Localization** (The date format must be as follows: month.date.year.)

1. Environmental needs (facilities, hardware, software, network, special tools)
   1. Test environment(s)

For performance testing a dedicated environment on VM and optional database. The configurations of the servers should be as much as close to production’s ones.

* 1. Testing tools

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Module** | **Software** | **Distribution** | **Where to install** | **Supported OS** | **Description** |
| CI | Jenkins | Inside docker image/Manual installation | In VM | Linux/Windows | The module that starts and controls the execution of your performance tests should be connected and authorized on your Jenkins server. |
| Monitoring agent | Performance Monitor/Telegraf | Custom installation package | In VM | Linux/Windows | The module gets all performance and health metrics from the host where it is installed (including Load Generator) to get a more accurate picture of your performance test's impact. |
| Load Generator | JMeter | On a work computer with VM | In VM | Linux/Windows | The module generates a load on an application (Server-side). |
| Visualizer | Performance Monitor/Grafana | Inside docker image | In VM | Linux/Windows | The module visualizes metrics that are collected by monitoring tools and generated by the load generator |
| Database | InfluxDB | Inside docker image | In VM | Linux/Windows | The module stores performance metrics that are collected by monitoring agents |

1. Test data

To have whole cycle of performance testing test data for TheBlog should be:

1. Reusable
2. Generated in necessary amount for different stage of testing at any time:
3. Cleanable (for example history, log records, generated data)
4. Performance Entry, Exit, and Suspension Criteria
   1. Entry Criteria

* Test plan is complete and approved by the mentor.
* Correct version is installed in performance testing environment, i.e. the version previously functionally tested and fixed if needed
* Test data is complete and in the performance testing environment in sufficient time to allow test scripts to be completed.
* Test accounts have been created in the performance testing environment in sufficient time to allow test scripts to be completed.
* Test scripts complete.
* All assigned resources are available to monitor the test.
  1. Exit Criteria
* All test scripts completed successfully
* No critical problems encountered
* All non-critical problems are logged
* All test logs are captured
* All post-test notifications sent
  1. Suspension Criteria
* Not all test scripts will complete
* Critical problems are encountered and logged
* Hardware errors prevent the completion of the test

1. Responsibilities

Good Guy 1 – Performance Analyst (mentor) from the performance program that checks the work of the tester

Good Guy 2 – Tester who works on the implementation of all program tasks

1. Schedule (test milestones and item transmittal events)

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Stage** | **Period** | **Notes** |
|  | Test strategy design |  |  |
|  | Test plan creation |  |  |
|  | Setting up test environment |  |  |
|  | Test data preparation |  |  |
|  | Script/Scenarios design |  |  |
|  | Practice: Task 1 running from Performance Testing Mentoring Program 2022 Q1Q3 |  |  |
|  | Practice: Task 2 running from PTMP 2022 |  |  |
|  | Practice: Task 3 running from PTMP 2022 |  |  |
|  | Practice: Task 4 running from PTMP 2022 |  |  |
|  | Practice: Task 5 running from PTMP 2022 |  |  |
|  | Practice: Task 6 running from PTMP 2022 |  |  |
|  | Practice: Task 7 running from PTMP 2022 |  |  |
|  | Practice: Task 8 running from PTMP 2022 |  |  |
|  | Practice: Task 9 running from PTMP 2022 |  |  |
|  | Practice: Task 10 running from PTMP 2022 |  |  |
|  | Practice: Task 11 running from PTMP 2022 |  |  |
|  | Practice: Task 12 running from PTMP 2022 |  |  |
|  | Practice: Task 13 running from PTMP 2022 |  |  |

1. Risks and contingencies
   1. Risks

* A significant difference in configuration from the production environment
* Performance testing results can be essentially different even in case of minor difference in think times, arrival rate and test duration
* During the execution of the tests, some major performance or functional problems that may require code changes, creation of a new build may be discovered and in that case it may be necessary to repeat the load test from the beginning
* Load test should be performed against a build that is solid enough, and that has been functionally tested, after code is complete. Failure to follow this rule may result on rework to update test scripts for every new build, plus the load test may need to be repeated from the beginning. This will affect the schedule
* Performance testing tool is not capable of identically reproducing real life scenarios - so results could only be trusted as having limited reliability level
* Network/systems latency issues
* Environment’s unavailability