# Performance Testing Strategy

The purpose of this document is to provide high-level description of the implementation of Animeal application performance testing strategy.  
The strategy includes general approaches to building performance testing infrastructure using loading and monitoring tools, and includes risks, metrics, and components to be tested.

* Objectives
* Test types
* Metrics
* Non-functional Requirements
* Testing environment
* Performance Testing Prerequisites
* Reporting
* Risks
* Test plans

# Objectives

The aim of performance activities is to test server-side performance of the application. The main focus is to highlight low-performance problems by analyzing server-side metrics.

The main performance testing objectives are:

* Measure response times on each test step and check if they comply to NFR.
* Find system resources bottlenecks by analyzing resources consumption (CPU and memory utilizations, network resources etc.) under given conditions.
* Provide analysis data to define a configuration that could offer optimum performance.

# Test types

During performance testing next types of tests are to be conducted:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test type** | **Description** | **Goal** | **Estimated time** |
| **Ramp-up** | Load starts with a small number of virtual users and gradually increases. Test is stopped when saturation point (capacity) is reached.  Should be performed as one of first main tests and after significant changes in the application or its configuration. | To find system capacity – the number of virtual users which the application support in stable state. | 1-3 hours, depending on system capacity |
| **Load** | The test is conducted under expected level of load or load calculated as 80% of capacity.  Checking basic application performance metrics. Usually, it is running after finding the capacity of the system. | To assess the behavior of the system under specific level of load and compare it against target/previous release results. | 2 hours |
| **Stability** | The test is conducted under the load of 80% of capacity for a long period of time.  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. | To assess system stability during long-time load and find memory leaks. Check if backend background activities affect system performance. | Up to 48 hours |

Additional test types can be applied in specific cases.

# Metrics

The following metrics should be usually monitored and collected for regular tests by default:

**Application metrics** (from load generator):

* number of virtual users
* number of requests
* hits per second (throughput)
* average response time
* 90 percentile of response time
* max of response time
* errors number

**System metrics** (from application's components):

* CPU utilization (%)
* memory utilization (%)
* network traffic

**Business metrics**:

* number of transactions (if not matching with single requests)
* transactions per time unit

For various tests additional metrics can be added to the list.  
After setting up the monitoring process the mentioned list can be reviewed and updated.

# Non-functional Requirements

Main common NFR for entire system is stable work under any significant load at daytime operating hours (6.00-23.59).

For different subsystems and their components NFR can be various depending on the specifics of the components.  
Exact numbers should be provided in performance testing plans prepared for each component to be tested.

The main source of NFR is expectations from the developing teams, PO, PM, BA.

|  |  |  |
| --- | --- | --- |
| **NFR metric** | **Acceptable value** | **Details** |
| **Response time** | <2 seconds | Load times for each step should be below 2 seconds (green zone), between 2 and 5 seconds (yellow zone). If load time >5 it is red zone, bug is to be created. 90 percentile means that 90% of transaction response times should be less than established value. |
| **CPU** | <80% | CPU load should be under the critical threshold to ensure stable performance. |
| **Memory** | N/A | There should be no memory leakage and significant performance degradation. |

# Testing environment

* Testing tool – JMeter
* File storage – S3
* Monitoring tools – Amazon Cloud Watch
* Test environment – QA Env
* Load generator environment – Amazon EC2

The load generator should be as close as possible to the application’s environment in topological and geographical terms.  
That significantly minimizes network latency (time to the first returned byte).

# Performance Testing Prerequisites

* Environment is stable to run performance tests, no other tests (processes, batch jobs etc.) are running at the same time.
* Test plan is complete/updated and approved by the client.
* Test users with previously defined roles are available.
* Test framework is completed according to updated performance tasks.
* Performance metrics are accessible for monitoring and collecting.

# Reporting

All the documents related to performance testing of the application are to be placed in the Confluence.

Documenting process includes:

* Collating performance testing strategy, test plans, performance related information to the Confluence.
* Adding test results to the Confluence.
* Preparation special analytical materials related to performance.
* Sending informational e-mail with the results of testing.

# Risks

* Performance testing in this scope covers only server-side processing.  
  It means that end-user response time will be increased by network time and client-side processing which differs for every user and cannot be fully predicted.
* Performance testing results may differ from production even in case of minor difference in think times, load distribution and test environment configuration.
* 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 of releasing the product.
* Performance testing tool is not capable of reproducing entirely realistic scenarios - so results could only be trusted as having limited reliability level.

# Test plans

Test plan is to be created for every separate test case. It should be based on the test strategy.

The cases to test are:

* Test Plan: Users

# Test Plan: Users

Performance testing approach, common aspects and goals are described in performance testing strategy.

* Scenarios
  + Admin
  + Volunteer
* Non-functional requirements

# Scenarios

### Admin

1. Manage feeding point (TBD)  
   Scenario distribution: 95%Steps:
   1. View the list of feeding point
   2. CRUD feeding point
2. Manage users(TBD)Scenario distribution: 5%  
   Steps:
   1. View the list of users
   2. View detailed info of user

### Volunteer

1. Find and view feeding point
2. Commit to fill a bowl
3. Upload a picture to mark the completion of the feeding

# Non-functional requirements

Baseline values will be defined and assessed during the first test iteration.

Common requirements:

|  |  |  |
| --- | --- | --- |
| **NFR metric** | **Acceptable value** | **Details** |
| **Expected level of load** | Admin: (TBD) (hits per second)  Volunteer: (TBD) (hits per second) | System should support maximum level of load expected to be on Production. |
| **Response time** | Admin: (TBD)  Volunteer: (TBD) | Load times for each step should be below 2 seconds (green zone), between 2 and 5 seconds (yellow zone). If load time >5 it is red zone, bug is to be created. 90 percentile means that 90% of transaction response times should be less than established value. |
| **CPU** | <80% | CPU load should be under the critical threshold to ensure stable performance. |
| **Memory** | N/A | There should be no memory leakage and significant performance degradation. |