Step #1 Identify performance acceptance criteria and metrics

Latency in seconds will be key metric for searching and browsing category.

Duration in seconds will be key metric for refreshing data source.

As we don't have any special requirements from the customer, assume that the product search should not take more than one seconds, the product browse - no more than two seconds.

We will count number of susses rates and the 85th percentile for latency.

Also identify performance thresholds - (fatal will be 0.5, warning will be 0,01).

Step #2 Identify key scenarios

We care only about searching and browsing, so there are only two possible scenarios:

Scenario	**Action**	**Input Data (queries for request)**	**Output Data (response body, headers)**	**Think Time (delay between invocations)**
Browsing the product catalog	Browse	Catalog tree	• Product description • Title • Category • Rating	4-15 seconds
Search for a specific product	Search	Product name	• Product description • Title • Category • Rating	2-5 seconds

Step #3 Identify type of testing

Performance testing. To verify response time of the application under normal (200 user actions per second - **load test#1**), moderate (400 user actions per second - **load test#2**) load and duration of data refreshing **(load test#1, load test #2, load test#3, load test #4, load test #5)**.

Load testing. To verify response time of the application under heavy (600 user actions per second - **load test#3**) load conditions. We can use results of this test to create baseline for identifying the change in performance in future.

Capacity test. Check the maximum number of requests per seconds that the application can handle before it crashes **(load test#4**). To introduce the customer with how his application is scalable.

Spike test. To check how system can handle moderate load and recover after this for several times. To be sure that eShop will be able to handle such kind of load in some sales hours, for example. **(load test#5)**.

Step #4 Strategy of testing

Let's assume that we have testing environment which are replica of our production environment, we can set up this environment for loading tests as quickly as possible using special tools. In this case, configuration of testing environment is not our concern, but we still need to manage testing data. We need to be sure that we have the same data in the system each time tests runs. For the begging we need to generate/create data sets, then we will make data refresh before load tests.

According to a lot of researches the most users are tend to start browsing over searching, so let's assume that we have N users on the site at the same time, 60% of them are going to browse, 40% - search.

We will validate successful code of response, output data type (JSON) and response headers to understand that we get correct response.

Load test #1

For the first load test we need to emulate 200 user actions per second. We take two threads, the first one includes 120 virtual users, the second - 80 users. First group has 15 seconds delay between invocations (think time), second group has 5 seconds delay between invocations. Second group starts with 10 seconds delay. They are working in parallel. They both sends request to the same global endpoint (host). They use different paths (to browseCategory, to searchPhrase) and

queries. There are two different data set (key-value pairs as queries) for browsing and searching.
Termination criteria for this load test are thirty minutes.

Load test #2

For the second load test we need to emulate 400 user actions per second. To make process of generation users easier we will use four threads: two includes 120 users, two - 80. First two groups have 15 seconds delay between invocations (think time), the third and the fourth groups have 5 seconds delay between invocations. Last two groups start with 10 seconds delay.
Termination criteria for this load test are thirty minutes.

Load test # 3

For the third load test we need to emulate 600 user actions per second. As in previous tests, we will share all users among six threads (of 120 and 80 users). Delay between invocations, delay in the beginning are the same.
Termination criteria for this load test are ten minutes.

Load test # 4

For the this test we don't have specific users actions per second. If previous test was okay, then we need make our load on eShop even more, so we will start from 600 user actions per second. We will have threads from load test #3, then we are gonna add one user per 10 seconds to every thread.
Termination criteria - when more than fifty percent of responses from the eShop aren't successful.

Load test # 5

In this load test we have four thread: a,b - 120 users for browsing, c,d - 80 users for searching. Thread a, b 15 seconds delay between invocations; c and d - 5 seconds delay between invocations.
A and c are working in parallel for 3 minutes, then with 180 seconds delay b and d start to work (try to increase load rapidly), there are four threads are for 3 minutes, then we have only two threads again (gradually reduce the load back).
Termination criteria for a,c - 540 seconds, for b,d - 180 seconds and with 180 seconds delay in the beginning.

Step #5 Report

Report for every test should include: number of susses rates and the 85th percentile for latency, results of validation, decision making (fatal, warning, ok) for key metrics according to the current baseline.