**Project Name -**  **cQube**

**Test Plan Document**

**June 2023**

**Version 5.0.3**

**Test Plan ID: cQube\_QA**

**Prepared By: Chetan**

**Revision Date: 01-June-2022**

**Approved By:**

**Approval Date:**

**TABLE OF CONTENTS**

1. [**Introduction**](#_u943y9umgfe6)3
2. **Test Items** 4

* [Features to be Tested](#_tyjcwt) 5
* [Features Not to be Tested](#_907tjzl8k4sc) 5
* [Test Approach](#_nmzclhr0ri52) 6
* [Test Pass / Fail Criteria](#_1t3h5sf) 6
* [Test Suspension / Resumption Criteria](#_17dp8vu) 7
* [Test Prerequisites](#_2s8eyo1) 7
* [Test Deliverables](#_d4q6k7in1la0) 7
* [Environment Needs](#_c91b0snzuzu9) 7
* [Roles And Responsibilities](#_bet5dsuj3kly) 7
* [Staffing / Training Needs](#_5r08ral9raka)8

1. [**Risk and mitigation**](#_26in1rg)8

* [Test Risks / Issues](#_lnxbz9) 8
* [Risks And Mitigation](#_35nkun2)9

1. **QA Criteria details** 9
2. **Api Automation Approach**  10
3. **Non functional Test Approach**  13
4. **Benchmarking Tool**  19
5. **Automation Tool Approaches**  20

Introduction

# 1.1 Project description

cQube Edu can be modeled as a class which will include - Event, Dimension, Dataset, Transformer, Indicator, Charts, Dashboard, Actions. The domain can be modified after instantiation using the inbuilt APIs. An example would look like [this](https://github.com/Samagra-Development/cQube_specs/blob/main/education_domain.json). The domain config is used by the starter script to initialize a cQube Ed instance..

# 1.2 Purpose of The Document

This document describes the plan for testing the cQube Application. This Test Plan document supports the following objectives:

* List the recommended test requirements at a high level
* Recommend and describe the testing strategies to be employed

# 1.3 References

* cQube-v5.0.3 Requirement design document
* cQube-v5.0.3 Technical document
* Sprint Task list and Technical document

2. Test Items

# 2.1 cQube QA Testing Flow Diagram



# 

# 

# 2.2 Features to be Tested For the Sprint

| **Test Type Activity** | **Features to be Tested** | **Test Schedule** |
| --- | --- | --- |
| **Installation Testing** | * One step installation testing. * Config files validation testing. * Verifying the application services(Postgres,Nifi,Kong,Node JS…) | Build is Released to QA - in this phase of testing verifying the both positive & negative scenarios. |
| **Functional Test**  **(Automation)** | * Spec , Ingestion process * API - Testing using Karate framework | Functional Testing of all the specs (schema) and Ingestion of input files using APIs |
| **Pre-Release Activitiy** | * End to End flow level test | After completion of functionality test - will be continuing with end to end to testing of cQube application |

# 

# 

# 

# 2.3 Features Not to be Tested

**< Need to update information >**

# 

# 

# 2.4 Test Approach

* Every feature will be tested as per the Project Timelines once the feature is made available for testing.
* Once build is Ready for QA then we will ready with test cases for the build , QA team will be discussed about their tasks and providing the effort list for the each tasks
* QA Team starting with smoke testing of below mentioned components of application
  + - One Step Installation
    - Data - API spec,Ingestion
* If any major , blocker defects found while doing smoke testing immediately build should be rejected from QA
* Creating Jira ticket and assigning to respective developers
* After Defects are cleared from smoke Test, initiating with functionality testing complete the application with positive and negative scenario
* Completion of both smoke and functional testing - updating the test execution result in to the each test cases result and preparing summary report
* Finally, once all the defects have been resolved successfully, we can start with system testing which covers end to end feasibility of the application
* Modifying the test cases, test result documents and upload to the repository

# 2.5 Test Pass / Fail Criteria

Code Coverage of about 70-80% & Pass percentage of test cases 95%+

Critical bugs: 0

# 

# 

# 

# 

# 2.6 Test Suspension / Resumption Criteria

* Testing of the feature should be suspended when any of the major functionality of the feature is not functional / working then testing should be suspended.
* Testing should be resumed when the feature is fully functional

# 2.7 Test Prerequisites

The Test Environment with the appropriate feature(s) to be tested should be ready prior to the Test Execution Phase. All the dependencies for testing feature(s) should also be made available with appropriate access.

# 2.8 Test Deliverables

* Test Summary and Test case execution Report - TC and TR
* Defect Status - can be updated on Jira
* Karate API automation TestCase document and Execution Reports
* Jmeter - Scenario document and Test Reports

# 2.9 Environment Needs

* AWS EC2 Machine - aws s3 configurations , azure configuration - 16 core machine (min required 16GB Ram)

# 2.10 Roles And Responsibilities

| **Role** | **Responsibility** |
| --- | --- |
| Test plan | Test Lead |
| Test Cases | QA Team |
| Test Execution Report | QA Team |

# 

# 2.11 Staffing / Training Needs

- Training needs in Karate framework used to test the API and JMeter.

# 2.12 Test code - repo management

* After completion of all the testing activities following QC documents will be pushed to the test folder which is created under a certain repository.
* API Automation Scripts
* WRK and Jmeter automation scripts
* Test Case and Test Result documents
* Test Execution Report

3. Risk and mitigation

# 3.1 Test Risks / Issues

Assumptions

* The feature(s) that are to be tested along with their dependencies will be made available by the development team on time

# 3.2 Risks And Mitigation

The following risks have been identified and the appropriate action identified to mitigate their impact on the project. The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered. The trigger is what milestone or event would cause the risk to become an issue to be dealt with.

**Note:** This one is applicable when new requirements are added to existing sprints.

| **Risk** | **Impact** | **Trigger** | **Mitigation Plan** |
| --- | --- | --- | --- |
| Scope Creep - Changes to the functionality may negate the test cases already written and also the effort on testing | High | CR raised, Functionality Changes, Delays in implementation | Add additional test members to complete the testing. However, this depends on the requirement change |

4.QA Criteria Details

# 4.1 Entry Criteria for QA

* Requirements are defined and approved.
* Availability of sufficient and desired test data.
* Test cases are developed and ready.
* Test environment has been set-up and all other necessary resources such as tools and devices are available
* Development phase/process provides useful information pertaining to software, its design, functionalities, structure, and other relevant features, which offer assistance in deciding the accurate entry criteria like functional and technical requirement, system design, etc.
* From testing phase, following inputs are considered:
  + Test Plan.
  + Test Strategy.
  + Test data and testing tools.
  + Test Environment.

# 

# 

# 4.2 Exit Criteria for QA

* Deadlines meet.
* Execution of all test cases.
* Desired and sufficient coverage of the requirements and functionalities under the test.
* All the identified defects are corrected and closed.
* No high priority or severity or critical bug has been left out.

5. API Automation Approach

Using the karate framework - will be able to do the API automation for the cQube spec and ingestion.

# 5.1 Required SET-UP configurations:

Follow the steps to set up the karate framework setup

* Java(8+), IDE, Maven dependencies in pom.xml
* Open Eclipse
* File > New > Maven project > Click on create a simple project > Next > Group Id: com.karate.com>Artifactid: karate>Finish
* Click on pom.xml
* Add dependencies within the project
* Add Maven dependencies in pom.xml (<https://mvnrepository.com/artifact/com.intuit.karate>)
* Karate core - compiling the feature files

- Karate Apache - Interface between API to server

- Karate JUnit4 - logs and reports generation

# 

# 

# 

# 

# 5.2 Process of API Automation:

* Once the framework is created. we have to create .feature (cucumber files) in the src/test/java package which is used to write the API scripts in the .feature file
* In the created .feature file we have to follow up gherkins syntax to get API response and add the validation assertions.

< – Gherkins Syntax – >

**Feature**: Keyword explains the **name of the feature** we are testing

**Background**: Prerequisite section - define url, json paths

**Scenario**:

**Given** URL

**When** method <http method>

**Then** match <status code>

**And** match < assertions based on response >

**Example: API Name:- cQube Spec -** /spec/event

**Feature:** Event Spec creation

**Background:** Define the url

Given url ‘application url’

\* def requestbody= read(‘json file location’)

**Scenario:** Validate the Spec/event creation

Given path ‘/spec/event’

And request requestbody

When method Post

Then assert responseStatus == 200

And match And match response != {}

And match responseType == 'json’

**Referring the above the example we will follow up for the all the api’s listed below**

* spec
* dimension
* schedule
* ingestion

# 5.3 Below are the listed possible API Response validations:

* Schema validation - body and response
* Response code and status name validation
* Data type validation of each response
* Null validation
* Format validation

# 5.4 Below are the listed Negative validations:

* Response code
* Invalid body syntax validation
* Data type validation - providing other data type
* Not null validations.
* Format validation - json , xml
* Error message validation - re triggering same API

# 5.5 Execution Approach:

* Some of scenarios are marked as @smoke , @functional which are comes under the test strategy
* creating the java file and adding the karate options and adding up tag name
* Running the specified test types based on giving parameters in command

**Commands:**

**mvn test -Dkarate.options="--tags @functional"**

* Once execution completes - Test Reports are stored in the location

**/target/karate-reports/karate-summary.html**

6. NON-FUNCTIONAL Test Approach

# 6.1 Purpose of Test Methodology:

The purpose of this section is to provide a high-level overview of the performance testing approach that should be followed for the cQube project. This must be presented to all the relevant stakeholders and should be discussed in order to gain consensus.

# 6.2 Introduction about Performance Test

As part of the delivery of the cQube, it is required that the solution meets the acceptance criteria, both in terms of functional and non-functional areas. The purpose of this document is to provide an outline for non-functional testing of the cQube solution.

This document covers the following:

* Entry and Exit Criteria
* Environmental Requirements
* Volume and Performance Testing Approach
* Performance Testing Activities

**Entry Criteria for Non functional**

The following work items should be completed/agreed upon beforehand in order to proceed with the actual performance testing activities:

* Non-functional test requirements document provided, with quantified NFRs where possible
* The critical use-cases should be functionally tested and without any critical bugs outstanding
* Design Architectural Diagrams approved and available
* Key use-cases have been defined and scoped
* Performance test types agreed
* Load injectors setup
* Any data setup needed - e.g. Appropriate number of users created in <DATASTORE>

**Exit Criteria for non functional:**

The performance testing activity will be completed when:

* The NFR targets have been met and performance test results have been presented to the team and approved.

# 6.3 Environmental Requirements

The performance tests will be run against a stable version of the cQube solution. and performed on a dedicated production-like environment (pre-production) assigned for performance testing with no deployments on that environment during the course of the performance testing.

**1. Load Injectors**

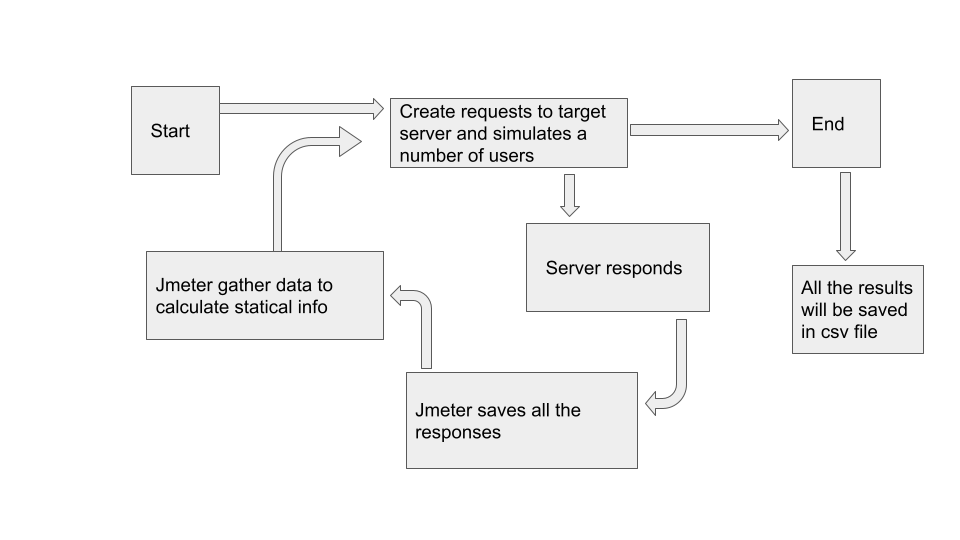
There will be one or more dedicated “load injectors” set up to initiate the required load for performance testing. The load injector could be a VM or multiple VMs that have an instance of JMeter running, initiating the requests.

**2. Test Tools**

Test tools used for Volume and Performance testing will be

**2.1 JMeter**

An open-source load testing tool. Predominantly used for volume and performance testing.



**Performance Test Plan using Jmeter**

* Start JMeter. Open the JMeter window in the terminal by clicking on cd Downloads/apache-jmeter-5.5/bin/jmeter.sh
* Rename the Test Plan. Change the name of the test plan node to Sample Test in the Name text box
* Add Thread Group.
* Add http request→url , path , port number, body
* Add Sampler
* Add Listener
* Run the Test Plan.
* View results in, summary report and View result tree listeners
* View the Output in CSV file

**2.2 Volume and Performance Testing Approach**

The cQube solution should be performant enough to manage the following load criteria. N.B. The numbers in the following table are for sample only - real values should be inserted once finalized by <CLIENT> NFR document.

**2.3 Number of Users**

Performance testing will run with a maximum of 1000[?] users. The users will be created in <DATASTORE> beforehand and be accessible via cQube Login API. Each request will login with a different userID.

**2.4 Assertions**

JMeter tool will be used to execute performance testing scripts. Within the scripts, there will be assertions stated to check for the above metrics as well as some basic functional checks to ensure correct responses are received for each request.

**2.5 Load Profiles**

The load profiles should be designed to mimic a typical average day’s traffic to <CLIENT> site. Please note that the traffic is only apportioned and limited to the Customer Identity and Access Management part of the site, i.e.

* Login
* Register
* Reset Password
* Forgot Password

**3 Load Testing**

After the baseline metrics are gathered, then the same simulation, which simulates a load profile, is run with an increased number of users to test against the target volumes. The idea of this load test is to test the system against a typical day’s load, simulating the ramp-ups, day’s peaks, and ramp-downs.

**4. Stress Testing**

The aim of stress testing is to find the breaking point of the system, i.e. at what point does the system become unresponsive. If auto-scaling is in place, the stress test will also be a good indicator at which point the system scales and new resources are added. For stress testing, the same simulation used for load testing is used but with a higher than expected load

**Performance testing activities**

The following activities are suggested to take place in order, to complete Performance Testing:

**Performance Test Environment Build**

* The load injectors should have enough capacity and should be managed remotely. Also, the location of the injectors should be agreed
* Real-time monitoring and alerting mechanisms should be in place and should cover the application, the servers as well as the load injectors.
* Application logs should be accessible.

**Use-Case Scripting**

* The performance testing tool which will be used is JMeter
* Any data requirements have been discussed for the use-cases to be scripted

**Test Scenario Build**

* The type of the test to be executed (Load/Stress etc)
* The load profile model should be agreed for each test type (ramp-up/down, steps etc)
* Incorporate think time into the scenarios

**Test Execution and Analysis**

The following tests should be executed in the following order:

* Load Test
* Stress Test
* Volume Test

Ideally, 3 Test runs of each test type will be performed. After each test run the application might be fine-tuned in order to increase its performance and then another test cycle will commence.

**Post-Test Analysis and Reporting**

* Capture and back up all the relevant data reports and archives.
* Determine the success or failure by comparing the test results to the performance targets. If the targets are not met then the appropriate changes should be made and then another test execution cycle will commence. It is unknown how many execution cycles will be needed in order to meet the agreed targets.
* Document and present the test results to the team.

**How we are planning to implement the performance testing in cQube**

* We are going to test each API with the users of 100,500,1000,3000,5000,10000,20000,30000,40000,50000
* With a ramp-up period of 1 sec and the average time should not cross more than 3000 milliseconds(3 sec)
* Final Report will be stored in csv file
* jmeter load testing test cases

<https://docs.google.com/spreadsheets/d/1od4uaW-65DTWSgbjrS1sW5Ll3i3EczVU4GKLAwW7C6g/edit?usp=sharing>

**How we are creating the JMeter scripts**

* Start JMeter. Open the JMeter window in the terminal by clicking on cd Downloads/apache-jmeter-5.5/bin/jmeter.sh
* Rename the Test Plan. Change the name of the test plan node to Sample Test in the Name text box
* Add Thread Group.
* Add HTTP request→provide the API url , path, port number, and body
* Add Listener - result tree , graph, view table report
* Run the Test Plan.
* View results in,summary report, and View result tree listeners
* Add assertions-->Response code assertion
* View the Output in CSV

7. Benchmarking Tool

**What is the WRK tool?**

WRK is a modern HTTP benchmarking tool capable of generating significant load when run on a single multi-core CPU.

**How to set up wrk tool?**

* Install wrk tool by using this command
* git clone --depth=1 https://github.com/wg/wrk.git
* cd wrk
* make -j
* The generated wrk executable is under this folder. This is how we use wrk for GET request benchmark:

**wrk -t 6 -c 200 -d 30s --latency https://google.com**

Some of the command flags for wrk

* -c: the number of connections to use
* -t: the number of threads to use
* -d: the test duration, e.g., 60s
* -s: the lua script to use for load testing our service (will cover in later section)
* --timeout how many seconds to timeout a request
* --latency: show the latency distribution for all the requests

8. Automation Tools Approaches for cqube 5.x

| **Feature** | **Automation Tools** |
| --- | --- |
| Installation |  |
| cQube spec and ingestion API | Karate Framework |
| cQube Dashboard | Selenium with Pytest framework |
| Performance Test | Jmeter and wrk |