# Software Quality Assurance Course Project Final Report

Matthew Lake (2716928)

April 21st 2025

Table of Contents

[Software Quality Assurance Course Project Final Report 1](#_Toc196132322)

[1.0 Project Description 3](#_Toc196132323)

[2.0 Functions Tested 3](#_Toc196132324)

[2.1 Initializing an OpenSearch Cluster Client and Testing with Ping 3](#_Toc196132325)

[2.2 Creating a Cluster Index 3](#_Toc196132326)

[2.3 Modifying a Cluster Index 4](#_Toc196132327)

[2.4 Deleting a Cluster Index 4](#_Toc196132328)

[2.5 Reindexing a Cluster Index 4](#_Toc196132329)

[2.6 Creating an Index Alias 4](#_Toc196132330)

[2.7 Updating an Index Alias 4](#_Toc196132331)

[2.8 Deleting an Index Alias 4](#_Toc196132332)

[2.9 Creating and Indexing a Document 4](#_Toc196132333)

[2.10 Bulk Document Indexing 4](#_Toc196132334)

[2.11 Retrieving a Document by its identifier 4](#_Toc196132335)

[2.12 Updating a Document 4](#_Toc196132336)

[2.13 Bulk Updating Documents 5](#_Toc196132337)

[2.14 Deleting a Document 5](#_Toc196132338)

[2.15 Searching for Documents 5](#_Toc196132339)

[2.16 Aggregating Documents 5](#_Toc196132340)

[3.0 Test Results 5](#_Toc196132341)

[4.0 Technologies Used 6](#_Toc196132342)

[5.0 Analysis 7](#_Toc196132343)

[6.0 Conclusion 7](#_Toc196132344)

## Project Description

For my software quality assurance course project, I tested the Python OpenSearch library called opensearch-py and indirectly OpenSearch itself. Here is a link to the opensearch-py GitHub page: <https://github.com/opensearch-project/opensearch-py>. The library is used for interacting with an OpenSearch cluster. The major functions of the library are communicating and facilitating index setup and management, document creation and management, and searching for and aggregating documents. The library can be thought as a helper and a wrapper around the OpenSearch APIs. OpenSearch itself is a document-based NoSQL database. OpenSearch databases are mainly used for search via full text search and application log analysis.

To carry out testing the opensearch-py library I created a Python automatic integration test suite. Here is a link to my project on GitHub: <https://github.com/Lake1824/matthew-lake-csu-cis-636-project-submission>. Steps for how to run the test suite are in the repository’s README.md file. I picked out a subset of functionalities to test and wrote a test suite for each one. In total I finished with fifty-two test cases. To test the various functionalities, I needed to spin up a test OpenSearch cluster to carry out the integration tests. I had that cluster in a docker container and another one for my Python test environment. I used to make commands to make running the test suite and cluster easier.

In addition to the automatic test suite, I experimented with and implemented basic Python lint checkers that are usually ran with the actual test cases. The linter libraries check for code formatting violations, types, and general coding bad practices. These are normally run over the entire application and tests. Finally, something I always wanted to try out is creating a basic ci/cd pipeline with GitHub Actions. I managed to have the test suite and linter checks run every time the repository was pushed too. Usually if a job like this fails you cannot merge your Pull Request into the main repository branch.

## 2.0 Functions Tested

I pulled out sixteen functions from the opensearch-py library to test for this project. For each functionality I setup different test cases either testing their successful path or their various unsuccessful paths. I setup different scenarios with the test OpenSearch cluster and then wiped the cluster after each test case. To setup scenarios I used test fixtures to reduce the amount of supporting code and functionalities needed for each test case.

### 2.1 Initializing an OpenSearch Cluster Client and Testing with Ping

For this function I tested the opensearch-py library’s ability to connect to a test OpenSearch cluster and verified it using the ping functionality. The cluster client is used to initiate and instruct an OpenSearch cluster to perform tasks like indexing documents or searching for documents. The corresponding Test IDs are 001 and 002.

### 2.2 Creating a Cluster Index

For this function I tested the opensearch-py library’s ability to create a new cluster index. An index is a collection of shards that hold documents. A cluster can be created with different configurations containing a JSON formatted list of settings and a JSON formatted index mapping. The corresponding Test IDs are 003 to 007.

### 2.3 Modifying a Cluster Index

For this function I tested the opensearch-py library’s ability to modify an existing cluster index’s mapping and settings. The corresponding Test IDs are 008 to 012.

### 2.4 Deleting a Cluster Index

For this function I tested the opensearch-py library’s ability to delete an existing cluster index. The corresponding Test IDs are 013 and 014.

### 2.5 Reindexing a Cluster Index

For this function I tested the opensearch-py library’s ability to reindex an existing cluster index. Reindexing entails creating a new cluster index and copying the existing documents over to the new index. I also tested filtering the index’s documents by using the OpenSearch query DSL interface. The corresponding Test IDs are 015 to 018.

### 2.6 Creating an Index Alias

For this function I tested the opensearch-py library’s ability to create a new cluster index alias. An index alias is used to hide or simplify an index’s name. The corresponding Test IDs are 019 and 020.

### 2.7 Updating an Index Alias

For this function I tested the opensearch-py library’s ability to update a cluster index alias. The corresponding Test IDs are 021 and 022.

### 2.8 Deleting an Index Alias

For this function I tested the opensearch-py library’s ability to deleting a cluster’s existing index alias. The corresponding Test IDs are 023 and 024.

### 2.9 Creating and Indexing a Document

For this function I tested the opensearch-py library’s ability to create a document using the DSL interface and index it into an existing cluster index. The corresponding Test IDs are 025 to 027.

### 2.10 Bulk Document Indexing

For this function I tested the opensearch-py library’s ability to index a collection of documents at once using the bulk indexing method. The corresponding Test IDs are 028 to 030.

### 2.11 Retrieving a Document by its identifier

For this function I tested the opensearch-py library’s ability to retrieve an existing document by its identifier(id). The corresponding Test IDs are 031 and 032.

### 2.12 Updating a Document

For this function I tested opensearch-py library’s ability to update an existing document. The corresponding Test IDs are 033 to 035.

### 2.13 Bulk Updating Documents

For this function I tested the opensearch-py library’s ability to update a collection of documents at once using the bulk updating method. This involves creating a script to send to the cluster for it to open and process the bulk update request. The corresponding Test IDs are 036 to 038.

### 2.14 Deleting a Document

For this function I tested the opensearch-py library’s ability to delete an existing document. The corresponding Test IDs are 039 and 040.

### 2.15 Searching for Documents

For this function I tested the opensearch-py library’s ability to create different search queries using the DSL interface, pass the queries to the cluster, and deserialize the response from the cluster. For testing this functionality, I used a set of different types of queries. The different queries I used during the tests are Full Text, Term Level, Boolean, and Geo. The corresponding Test IDs are 041 to 048.

### 2.16 Aggregating Documents

For this function I tested the opensearch-py library’s ability to create different aggregations using the DSL interface, pass the aggregation queries to the cluster, and deserialize the response from the cluster. For testing this functionality, I used Bucket and Metric aggregations in my tests. The corresponding Test IDs are 049 to 052.

## 3.0 Test Results

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Name** | **Results** |
| 001 | test\_client\_successfully\_pings\_cluster | Pass |
| 002 | test\_client\_cannot\_ping\_cluster\_due\_to\_missing\_cluster | Pass |
| 003 | test\_successfully\_creating\_an\_index | Pass |
| 004 | test\_unsuccessfully\_creating\_an\_index\_due\_to\_incorrectly\_formatted\_mapping | Pass |
| 005 | test\_unsuccessfully\_creating\_an\_index\_due\_to\_incorrect\_mapping\_type | Pass |
| 006 | test\_unsuccessfully\_creating\_an\_index\_due\_to\_incorrect\_setting | Pass |
| 007 | test\_unsuccessfully\_creating\_an\_index\_due\_to\_non\_existent\_cluster | Pass |
| 008 | test\_successfully\_modifying\_an\_index\_settings\_by\_adding\_a\_setting | Pass |
| 009 | test\_successfully\_modifying\_an\_index\_mappings\_by\_adding\_a\_field | Pass |
| 010 | test\_unsuccessfully\_modifying\_an\_index\_due\_to\_incorrect\_index\_setting | Pass |
| 011 | test\_unsuccessfully\_modifying\_an\_index\_due\_to\_incorrect\_mapping\_attribute\_type | Pass |
| 012 | test\_unsuccessfully\_modifying\_an\_index\_due\_to\_non\_existent\_cluster | Pass |
| 013 | test\_successfully\_deleting\_an\_index | Pass |
| 014 | test\_unsuccessfully\_deleting\_an\_index\_due\_to\_non\_existent\_cluster | Pass |
| 015 | test\_successfully\_reindexing\_an\_index\_with\_changing\_the\_index\_name | Pass |
| 016 | test\_successfully\_reindexing\_an\_index\_with\_only\_moving\_documents\_that\_have\_age\_greater\_than\_35 | Pass |
| 017 | test\_unsuccessfully\_reindexing\_an\_non\_existing\_index | Pass |
| 018 | test\_unsuccessfully\_reindexing\_an\_index\_with\_invalid\_query | Pass |
| 019 | test\_successfully\_creating\_an\_index\_alias\_for\_existing\_index | Pass |
| 020 | test\_unsuccessfully\_creating\_an\_index\_alias\_for\_an\_non\_existing\_index | Pass |
| 021 | test\_successfully\_updating\_an\_index\_alias | Pass |
| 022 | test\_unsuccessfully\_updating\_an\_index\_alias\_for\_an\_non\_existing\_index | Pass |
| 023 | test\_successfully\_deleting\_an\_index\_alias | Pass |
| 024 | test\_unsuccessfully\_deleting\_an\_index\_alias\_for\_an\_non\_existing\_index | Pass |
| 025 | test\_creating\_and\_indexing\_a\_document\_with\_dsl | Pass |
| 026 | test\_not\_creating\_and\_indexing\_a\_document\_with\_misspelled\_attribute\_name | Pass |
| 027 | test\_not\_creating\_and\_indexing\_a\_document\_due\_to\_non\_existent\_cluster | Pass |
| 028 | test\_creating\_and\_bulk\_indexing\_documents\_with\_dsl | Pass |
| 029 | test\_not\_creating\_and\_indexing\_a\_document\_with\_misspelled\_attribute\_name | Pass |
| 030 | test\_not\_creating\_and\_indexing\_a\_document\_due\_to\_non\_existent\_cluster | Pass |
| 031 | test\_successfully\_retrieving\_a\_document\_by\_its\_id | Pass |
| 032 | test\_not\_retrieving\_a\_document\_by\_its\_id\_due\_to\_non\_existent\_cluster | Pass |
| 033 | test\_updating\_a\_document\_with\_dsl | Pass |
| 034 | test\_not\_updating\_a\_document\_with\_incorrect\_id | Pass |
| 035 | test\_not\_updating\_a\_document\_due\_to\_non\_existent\_cluster | Pass |
| 036 | test\_successfully\_bulk\_updating\_documents | Pass |
| 037 | test\_not\_bulk\_updating\_a\_document\_due\_to\_incorrect\_script | Pass |
| 038 | test\_not\_bulk\_updating\_a\_document\_due\_to\_missing\_index | Pass |
| 039 | test\_successfully\_deleting\_a\_document | Pass |
| 040 | test\_not\_deleting\_a\_document\_due\_to\_non\_existent\_cluster | Pass |
| 041 | test\_full\_text\_query\_returns\_results | Pass |
| 042 | test\_term\_level\_query\_returns\_results | Pass |
| 043 | test\_compound\_bool\_query\_returns\_results | Pass |
| 044 | test\_geo\_query\_returns\_results | Pass |
| 045 | test\_general\_match\_all\_query\_returns\_results | Pass |
| 046 | test\_query\_returns\_no\_results | Pass |
| 047 | test\_searching\_a\_missing\_index\_raises\_transport\_error | Pass |
| 048 | test\_invalid\_query\_raises\_unknown\_dsl\_error | Pass |
| 049 | test\_bucket\_aggregation | Pass |
| 050 | test\_metric\_aggregation | Pass |
| 051 | test\_invalid\_aggregation\_raises\_unknown\_dsl\_error | Pass |
| 052 | test\_aggregating\_a\_missing\_index\_raises\_transport\_error | Pass |

## 4.0 Technologies Used

* Docker
* Python version 3.13
* Python Libraries
  + Pytest version 8.3.5
  + Black version 25.1.0
  + MyPy version 1.15.0
  + Flake8 version 7.2.0
  + Opensearch-py version 2.8.0
  + Poetry version 2.0.1
* Pycharm
* OpenSearch version 2.19.1
* Make
* Bash
* Github and Git
* Github Actions

## 5.0 Analysis

After developing and executing fifty-two test cases testing the various opensearch-py library’s core functionalities the results show that the libraries core functionalities are stable, functional, and reliable. Most importantly the functionalities demonstrate correctness and operate as intended. If I would continue testing opensearch-py and in turn OpenSearch I would conduct performance and stress testing. Also, testing the rest of the opensearch-py library functionalities.

## 6.0 Conclusion

During the course of this project, I developed and created fifty-two automatic integration test cases testing the opensearch-py library and indirectly OpenSearch itself. I setup Docker containers for the testing environment and execution and another container for a test OpenSearch cluster. I used the pytest library as a testing framework. I experimented and implemented linter libraries and checks. Finally, I experimented and implemented a GitHub Action job for executing the test suite in a GitHub runner.

I learned a few things during this project. The first thing I learned was the formal process of quality assurance testing and planning. Second, was using the opensearch-py and other Python libraries. The final thing I learned was setting up a GitHub Action job. I really enjoyed this project and would recommend others do it to get a better understanding of quality assurance testing in general.