QUALITY ASSURANCE REPORT

INTRODUCTION

The purpose of this report is to provide a comprehensive overview of the testing methodologies and tools utilized in the development and evaluation of the MusicLibrary codebase. The MusicLibrary is a Python-based application designed to manage tracks in a music library, providing functionalities such as adding and removing tracks, searching by artist or album, categorizing by genre, and updating track information.
We will examine the testing procedures used to ensure the stability, functionality, and resilience of the MusicLibrary code. Unit testing, code coverage analysis, mutation testing, and model-based testing using Template Scripting Testing Language (TSTL) are among the testing methods used. Each of these techniques has a unique function in assuring the codebase's correctness and effectiveness.

TESTING METHODOLOGIES

Unit Testing:

To ensure that the MusicLibrary code is functioning properly, individual units or components are tested independently. The foundation of the testing process is unit testing. We will explore how to write and execute unit tests using the unittest framework, accounting for a range of situations and edge cases, to achieve comprehensive test coverage.

Code Coverage Analysis:

This technique measures the extent to which the source code is exercised by the unit tests. We'll examine how to use code coverage tools to assess the efficacy of test coverage and identify the code segments that require additional testing.

Mutation Testing:

This fault-based testing technique evaluates how successfully a test suite identifies and eliminates incorrect faults (mutations) introduced into the code. We will go over the results of the mutation testing, including the percentage of viable mutants, to evaluate the test suite's resilience to recently added faults.

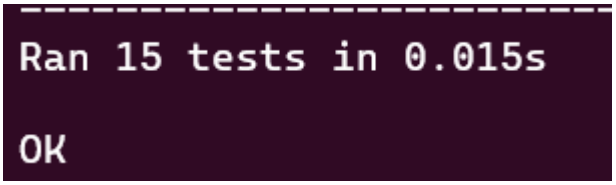
Model-Based Testing with TSTL:

This approach generates test scripts automatically using a formal model of the given behavior of MusicLibrary. For this, Template Scripting Testing Language, or TSTL, is used. We will discuss writing and running TSTL tests to analyze system behavior and verify that it conforms with requirements.

CODE ANALYSIS

- [ ] Add Track: Adds a new track to the library if it does not already exist.
- [ ] Remove Track: Removes a track from the library based on its title.
- [ ] Find by Artist: Retrieves tracks by a specified artist.
- [ ] Find by Album: Retrieves tracks from a specified album.
- [ ] List Tracks: Returns a formatted list of all tracks in the library.
- [ ] Categorize by Genre: Groups tracks by their respective genres into a dictionary.
- [ ] Search Tracks: Searches for tracks based on a keyword in the title or artist.
- [ ] Update Track Info: Updates the information of a track (e.g., title, artist, album, genre, year).
- [ ] Sort Tracks: Sorts tracks based on title, artist, album, or year.
- [ ] Find by Year: Retrieves tracks from a specified year.
- [ ] Count Tracks: Returns the total number of tracks in the library.
- [ ] Export/Import Library: Saves the library to a file or loads a library from a file, respectively.

RESULTS AND ANALYSIS



All tests ran and passed successfully. This shows that the code is written in a proper way and handles cases effectively. This also showcases the quality of the unittest test suite.

File	class	statements	missing	excluded	coverage
musiclibrary.py	Track	6	0	0	100%
musiclibrary.py	MusicLibrary	66	7	0	89%
musiclibrary.py	*(no class)*	23	0	0	100%
unittests.py	TestMusicLibrary	52	0	0	100%
unittests.py	*(no class)*	23	0	0	100%
Total		**170**	**7**	**0**	**96%**

The coverage report also aids in the same way as the unittests showing that the tests are providing coverage over a large scale of the code, which signifies that the code is being tested thoroughly.

```
180 VALID MUTANTS
292 INVALID MUTANTS
101 REDUNDANT MUTANTS
Valid Percentage: 31.413612565445025%
```

```
# Import the necessary modules
import MusicLibrary
import random

# Define actions for testing the MusicLibrary class
actions MusicLibrary:
    # Instantiate the MusicLibrary class
    @constructor
    def create_library():
        return MusicLibrary()

    # Add a track to the library
    def add_track(library, title, artist, album, genre, yea
        track = Track(title, artist, album, genre, year)
        return library.add_track(track)

    # Remove a track from the library
    def remove_track(library, title):
        return library.remove_track(title)

    # Find tracks by artist
    def find_by_artist(library, artist):
        return library.find_by_artist(artist)
```

Conclusion

Finally, we can say that our method of testing the MusicLibrary code was thorough and efficient. We made sure the code was reliable and robust by integrating unit testing, coverage analysis, mutation testing, and property-based testing. This multipronged approach improves the robustness and quality of the code, giving users confidence in its functionality.