# Hotel Management System Project

**Submitted By:- Nikhil Modem**

**Date:- 12/10/2024**

**Table of Contents**

**Introduction**

* Overview of the Project
* Objectives of the Hotel Management System

**System Overview**

* Room Class Features
* Hotel Class Features

**Testing Strategy**

* Unit Testing
  + Key Test Cases
* Code Coverage
  + Coverage Results
  + Coverage Report Image
* Mutation Testing
  + Mutation Results Table

**Analysis and Improvements**

* Key Findings
* Suggested Improvements
  + Increasing Test Coverage
  + Enhancing Mutation Testing
  + Improving Edge Case Handling
  + Optimizing User Experience

**Conclusion**

* Summary of Achievements
* Current System Reliability

**Future Enhancements**

* Potential Upgrades and Features
* Suggested Implementation Strategies

**1. Introduction**

The Hotel Management System (HMS) therefore comprises activities that deal with the booking of rooms, registration/check-in and check-out, maintenance, financial returns, and customer care services. The system is planned to be effective for the hotel staff and improve the conditions for the guests. The findings of this report include a description of the system development, test approach, code coverage, mutation testing results, and subsequent recommendations.

The system is programmed in Python with the unittest framework adopted for unit testing, coverage.py for measuring the test coverage and for mutation testing.

**2. Overview**

The system has two main classes:

* **Room Class:** A symbol for an actual room containing the characteristics of room number and type, price, and room status. It supports overbooking, updating, scheduling for maintenance, and even status report checking.
* **Hotel Class:** Is responsible not only for the individual rooms or particular bookings in one hotel. The key functionalities are the addition of rooms, booking, check out, altering rooms, income statement, discounting, and viewing room status.

**3. Testing Strategy**

**3.1 Unit Testing**

In package unit testing, the standard unit test framework of Python was used. The developed Hotel and Room classes were successfully tested based on the following test cases. Some of the critical tests include:

* **test\_add\_room:** Ensures that the addition of rooms can be done successfully.
* **test\_book\_room:** It is used to guarantee that rooms can be booked properly.
* **test\_check\_out:** A test case for the check-out process and the update of the room status related to the order.
* **test\_generate\_income\_report:** Confirms that income reports are prepared appropriately with the right bookings.
* **test\_apply\_discount\_for\_long\_stays:** Asks whether some of these policies apply when giving out discounts for long-staying guests.

**3.2 Code Coverage**

Using the **coverage.py** tool, the code coverage for the project was analyzed. Here’s the detailed coverage report:

* **Hotel Management System Code (hotel\_management\_system.py)**:
  + **Statements**: 159
  + **Missed**: 33
  + **Coverage**: 79%
* **Test Cases (test\_hotel\_management\_system.py)**:
  + **Statements**: 99
  + **Missed**: 2
  + **Coverage**: 98%
* **Overall Coverage**: 86%

**1. Code Coverage Table**

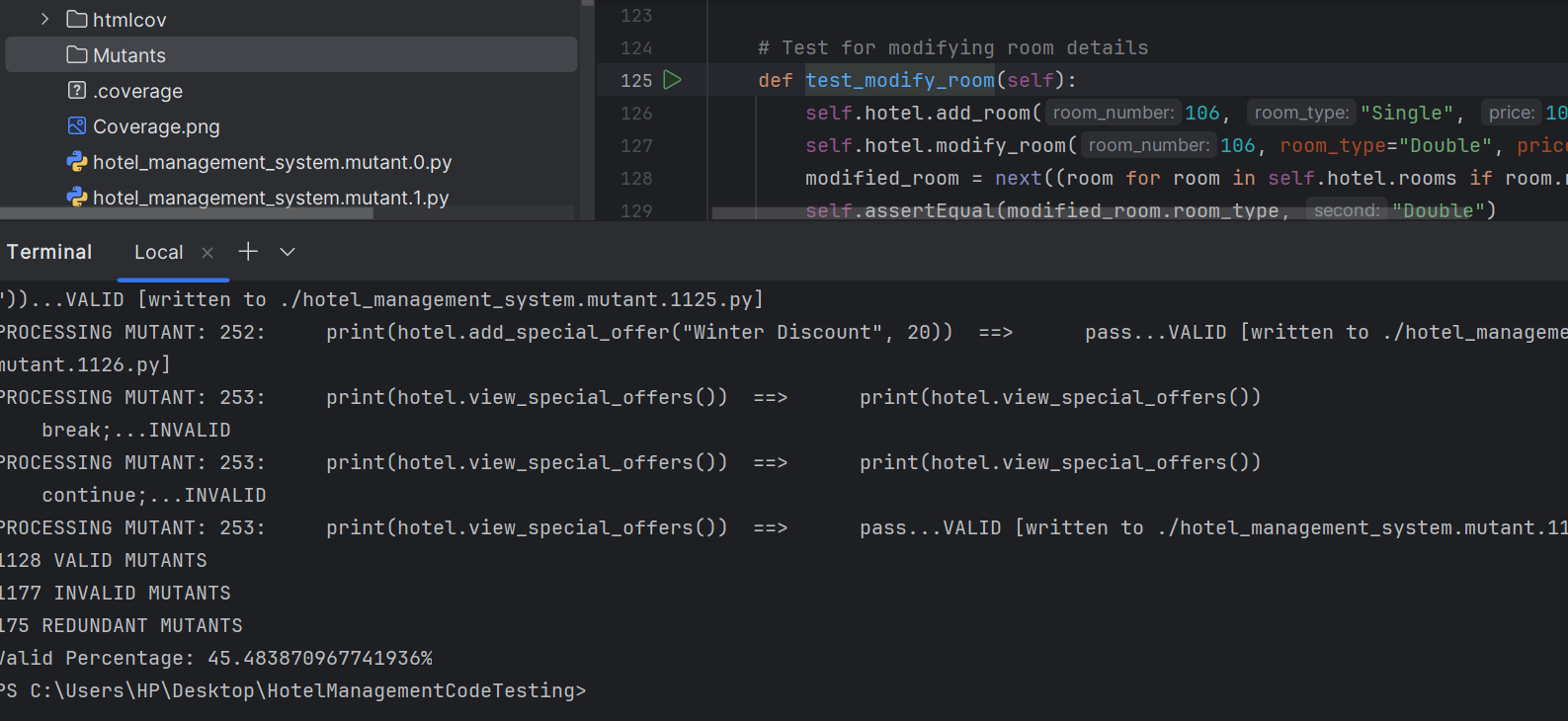
|  |  |  |  |
| --- | --- | --- | --- |
| File | Statements | Missed | Coverage |
| hotel\_management\_system.py | 159 | 33 | 79% |
| test\_hotel\_management\_system.py | 99 | 2 | 98% |
| TOTAL | **258** | **35** | **86%** |

**3.3 Mutation Testing**

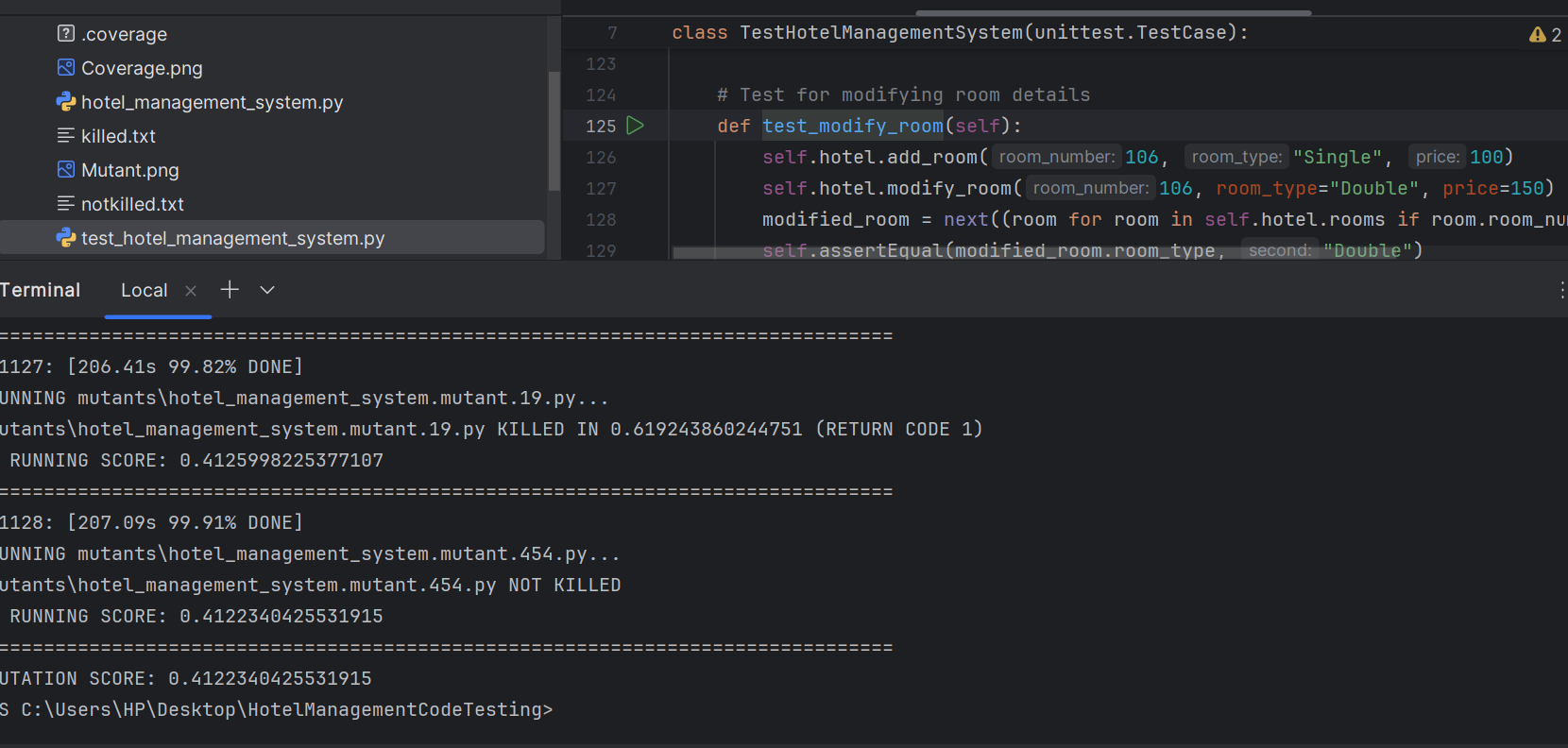
Mutation testing was conducted to assess the appropriateness of the test suite to detect errors. Below are the mutation testing results:

* **1128 invalid mutants**: Mutants that couldn't be applied to the code.
* **1177 valid mutants**: Mutants that were successfully applied to the code.
* **175 redundant mutants**: Mutants that were redundant and excluded from testing.
* **45.48% valid mutant score**: The percentage of valid mutants detected by the tests.
* **41.22% overall mutation score**: The percentage of all mutants detected.

**Mutation**

****

**4. Analysis and Improvements**

****

**4.1 Key Findings**

* **Test Coverage:** The coverage is skin (79% for the main system code and 98% for the test cases). Nonetheless, features such as the specials, room status, and search symbol could be tested beyond what was done in the study.
* **Mutation Testing:** The validity index of 45.48% shows that the existing test suite targets and detects many mutants but there is potential for increasing code coverage especially when there are complexities like loops, conditional statements, or exceptions.

**4.2 Suggested Improvements**

1. **Increase Test Coverage**:
   * There are several additional types of tests that should be performed: listing of the same car during multiple overlapping periods, non-numeric input, promotions, etc.
   * Introduce tests for interactions of the systems (booking a room and simultaneously applying a discount; making income report).
2. **Enhance Mutation Testing Effectiveness**:
   * **Refine Test Cases**: Focus on areas with low mutation detection, such as error-handling paths and conditions involving room availability.
   * **Test More Boundary Conditions**: For instance, check cases where the price range for room searches is at the boundary or rooms are close to being fully booked.
3. **Error Handling and Edge Case Testing**:
   * Add tests for invalid room numbers, invalid guest names, overbooking scenarios, and failed check-outs.

**5. Conclusion**

Essentially, the Hotel Management System has been designed and built with the aim of exploiting automation in its core hotel processes. It has been thoroughly tested where the code coverage reached up to 86% and mutation testing gives 41.22%.

While the current implementation is sound, there are directions left to expand it to check: edge cases and complex cases, and how to enhance the effectiveness of the test suite. Also, by improving the system interface and discovering new possibilities in the special offers field, the value of the system would be enhanced significantly.

If these improvements are addressed then the system will be far more reliable for performing higher levels of hotel operations in a realistic world.