# Testing report: Appointment Booking Management System

## 1) What did you test?

In this testing report, we focused on testing the "Appointment Booking System," which is a software system designed to manage user registrations and appointments. We conducted a series of tests to ensure the functionality and reliability of this system. We tested various aspects of the system's behavior, including user registration, appointment booking, slot availability management, and user-specific functionality.

Here is a breakdown of the key areas and functionalities that were tested:

User Registration: We tested the system's ability to register users with unique usernames and emails. This included both successful and unsuccessful registration attempts, checking for duplicate usernames.

Appointment Booking: We tested the system's ability to book appointments for users at specified dates and times. We checked for successful booking and verified that it correctly updated available slots. Additionally, we tested for scenarios where booking was not possible due to slot unavailability or user non-existence.

Confirmation and Reminder Emails: We tested the system's email-related functions, including sending confirmation and reminder emails to users. We verified that the email content was generated correctly and that users were appropriately notified.

Appointment Cancellation and Rescheduling: We tested the cancellation and rescheduling of appointments to ensure that booked appointments could be canceled, making the slot available again, and appointments could be rescheduled without issues.

Slot Availability Management: We tested the system's ability to add, remove, and check the availability of slots for booking.

User-Specific Functionality: We tested functions that were specific to users, such as retrieving their appointments and upcoming appointments.

## 2) How did you test it?

We employed a combination of manual unit tests and automated unit tests using the unittest framework. The unittests were written to test specific functionalities within the Appointment Booking System.

Manual Unit Tests: These tests were written by the testing team to target specific functions and scenarios within the system. Manual unit tests were used for functions like registering users, booking appointments, and checking slot availability.

Automated Unit Tests: We used the unittest framework to automate the testing of various system functions, such as sending emails, retrieving appointments, and managing available slots. These automated tests provided consistent and repeatable results.

## 3) How good were your tests, as measured in a few ways:

### a) Bugs Found:

During our testing process, we identified and reported several issues and bugs in the system, including:

1. Duplicate user registration was allowed, which is inconsistent with the expected behavior.
2. The reschedule appointment functionality did not handle some edge cases correctly.

### b) Code Coverage using coverage.py:

Using coverage.py, we assessed the code coverage achieved by our unit tests. The analysis indicated that our tests covered approximately 96% of the codebase. While this is a high coverage percentage, the uncovered 4% may contain untested edge cases or exceptional scenarios that could potentially lead to defects.

```
(myenv)                                                                 coverage run -m unittest tests.py

................
----------------------------------------------------------------------
Ran 16 tests in 0.005s

OK
(myenv)                                          Appointement_Booking_System % coverage report -m

Name       Stmts   Miss  Cover   Missing
----------------------------------------
app.py        91      4    96%   53, 66, 97, 110
tests.py      99      1    99%   122
----------------------------------------
TOTAL        190      5    97%
(myenv)                                                                 % coverage html

Wrote HTML report to htmlcov/index.html
```

## c) Run Mutants using Universal Mutator:

We utilized the Universal Mutator tool to generate mutants and evaluate the effectiveness of our tests in detecting faults. The results showed:

207 Valid Mutants
383 Invalid Mutants
172 Redundant Mutants
Valid Percentage: 27.17%

```
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
    continue;...INVALID
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      pass.
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 120:     def reschedule_appointment(self, old_date, old_time, new_date, new_time, user_name):  ==>      def r
PROCESSING MUTANT: 121:         if self.cancel_appointment(old_date, old_time):  ==>          if self.cancel_appointment(old_dat
        break;...INVALID
PROCESSING MUTANT: 121:         if self.cancel_appointment(old_date, old_time):  ==>          if self.cancel_appointment(old_dat
        continue;...INVALID
PROCESSING MUTANT: 121:         if self.cancel_appointment(old_date, old_time):  ==>          if self.cancel_appointment( old_ti
PROCESSING MUTANT: 121:         if self.cancel_appointment(old_date, old_time):  ==>          if not (self.cancel_appointment(ol
PROCESSING MUTANT: 121:         if self.cancel_appointment(old_date, old_time):  ==>          pass...INVALID
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              return self.bo
         break;...INVALID
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              return self.bo
         continue;...INVALID
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              return self.bo
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              return self.bo
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              return None...
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              pass...VALID [
PROCESSING MUTANT: 122:             return self.book_appointment(new_date, new_time, user_name)  ==>              return self.bo
PROCESSING MUTANT: 123:         return False  ==>          return False
        break;...INVALID
PROCESSING MUTANT: 123:         return False  ==>          return False
        continue;...INVALID
PROCESSING MUTANT: 123:         return False  ==>          return None...VALID [written to ./app.mutant.206.py]
PROCESSING MUTANT: 123:         return False  ==>          pass...REDUNDANT
207 VALID MUTANTS
383 INVALID MUTANTS
172 REDUNDANT MUTANTS
Valid Percentage: 27.165354330708663%
```

The low valid percentage suggests that our tests were not highly effective in identifying faults introduced by mutants. This indicates a potential area for improvement in our test suite.

In summary, our tests were successful in identifying some critical issues, achieved a high code coverage rate, but demonstrated room for improvement in terms of detecting mutants. Further refinement of the test suite and additional edge case testing could enhance the robustness of the system.