Test Plan

Assignment

# **DOCUMENT CONTROL**

|  |  |
| --- | --- |
| **Document Name** | Test Plan |
| **Reference Number** | - |
| **Version** | V1.0 |
| **Project Code** | UECS2354 Software Testing Assignment |
| **Status** | [to be updated] |
| **Date Released** | [to be updated] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Position** | **Signature** |
| Prepared By: |  |  |  |
| Reviewed By: |  |  |  |
| Approved By: |  |  |  |
| Approved By: |  |  |  |

# **VERSION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Release Date** | **Section** | **Amendments** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Content

[DOCUMENT CONTROL 1](#_Toc164875015)

[VERSION HISTORY 2](#_Toc164875016)

[1.0 INTRODUCTION 1](#_Toc164875017)

[1.1 Objectives 1](#_Toc164875018)

[1.2 Scope 1](#_Toc164875019)

[1.3 References 1](#_Toc164875020)

[2.0 TEST PLANS 2](#_Toc164875021)

[2.1 Test Items 2](#_Toc164875022)

[2.2 Features to be tested 4](#_Toc164875023)

[2.3 Features Not To Be Tested 6](#_Toc164875024)

[2.4 Test Basis 6](#_Toc164875025)

[2.5 Test Conditions 7](#_Toc164875026)

[2.6 Entry Criteria 9](#_Toc164875027)

# **1.0** **INTRODUCTION**

## 1.1 Objectives

The test plan supports the following objectives:

1. To verify the validity of users, including username, user member and user exclusive reward.
2. To verify that the system properly handles the exclusive rewards for normal member in room allocation.
3. To identify the quantity of rooms allocated to the user to verify the system's accuracy in allocating rooms based on user type and availability.
4. To verify that the system successfully places the users into the waiting list based on their member type when their booking request cannot be fulfilled.
5. To verify that the system accurately updates the number of rooms for all room types after every cancellation of booking.
6. To verify that the system successfully cancels the booking in waiting list and remove the users from the waiting list based on their member type when necessary.
7. To verify the integration between Booking, Waiting List, User and Room classes.
8. To verify that the system handles the error properly.

## 1.2 Scope

The scope of testing only covers the functionalities in the **Hotel Room Booking System**. This includes allocation of rooms, management of bookings and waiting lists, cancellation of bookings as well as handling of exclusive rewards. This test plan would cover integration test and unit test using Mockito, parameterised test and boundary value analysis. It would ensure the system meets the specified requirements and suitable for its intended objectives.

## 1.3 References

N/A

# **2.0** **TEST PLANS**

## 2.1 Test Items

1. **User class**

* *Constructor*

1. public **User** (String *name*, String *member\_type*, boolean *excl\_reward*)

* This constructor throws an exception if the name and member type is empty or null. Moreover, if *member\_type* is not equal to “VIP”, “member”, or “normal”, an exception will be thrown as well. “VIP” means VIP member, “member” means ordinary member, and “normal” means non-member. In addition, the system will throw an exception when exclusive reward (*excl\_reward*) is attempted to be assigned to a "normal" user. Lastly, if the exclusive reward is not Boolean value, an exception will be thrown.
* *Getter Method*

1. public String **getName** ()
2. public String **get\_member\_type** ()
3. public boolean **get\_excl\_reward** ()

* *Setter Method*

1. public void **set\_excl\_reward** (boolean *excl\_reward*)

* Within this setter method, the system verifies whether exclusive reward (*excl\_reward*) is attempted to be assigned to a "normal" user. If this is the case, an exception is thrown. Besides, if the exclusive reward (*excl\_reward*) is not in Boolean value, exception will be thrown.

1. **Booking class**

* *Constructor*

1. public **Booking** (User *user,* int *vip\_rooms*, int *deluxe\_rooms,* int *standard\_rooms*)

* *Getter Method*

1. public User **getUser** ()
2. public int **get\_vip\_rooms** ()
3. public int **get\_deluxe\_rooms** ()
4. public int **get\_standard\_rooms** ()

* *Setter Method*

1. public void **setBooking** (Room *room*, WaitingList *waiting\_list*, int *requested\_room\_qty)*

* Initially, if *requested\_room\_qty* is less than or equal to zero, the system will throw an exception.
* The booking is processed based on *member\_type*.
* For each type of member, the system will first check whether the number of rooms requested exceeds the maximum number of rooms allowed to be booked. If this is the case, an exception is thrown.
* For “VIP” users, the system will prioritize “VIP” rooms. If there is no “VIP” room, a Deluxe room will be arranged; If there is no “Deluxe” room either, a “Standard” room will be arranged.
* For “member” users, the system will give priority to “Deluxe” rooms. If the “Deluxe” room is not available, a “Standard” room will be allocated. However, if there are no “Standard” rooms, the system will use exclusive reward (*excl\_reward*), if any, to allocate “VIP” rooms, else, user will be placed into waiting list.
* For “normal” users, the system only allocates “Standard” rooms.
* If the user's request cannot be fulfilled, the *user* will be placed into waiting list based on user's member type.
* If the booking is successful, the allocated room quantity will be updated.
* *Method*

1. public void **cancelBooking** (Room *room*, WaitingList *waiting\_list*)

* This method will either cancels the booking and releases the booked room or removes user from waiting list if applicable.
* The system searches the user through *waiting\_list*, and if a matching user is found, it removes the user from *waiting\_list*.
* If the user is not in *waiting\_list*, the *user* is assumed to have a successful booking, so the system updates the *room* quantity by adding back the previously booked room.

1. **Room class**

* *Constructor*

1. public **Room** (int *vip\_rooms*, int *deluxe\_rooms,* int *standard\_rooms)*

* *Getter Method*

1. public int **get\_VIP** ()
2. public int **get\_deluxe** ()
3. public int **get\_standard** ()

* *Setter Method*

1. public void **set\_VIP** (int *num*)
2. public void **set\_deluxe** (int *num*)
3. public void **set\_standard** (int *num*)

* *Method*

1. public boolean **checkRoom** (String *room\_type)*
2. **Printer class**

* *Method*

1. public void **printInfo** (Booking *booking*)
2. **WaitingList class**

* Constructor

1. public **WaitingList()**

* This constructor initialise the array lists for the waiting list of “VIP” user, “member” user and “normal” user.
* Getter Method

1. public ArrayList<User> **getWaiting(**String*member****\_****type***)**

* Method

1. public void **addWaiting(**Booking*booking***)**
   * This method will first retrieve the user from the booking object and get the user's member type. Then, the user is added into waiting list based on their member type.
2. public void **removeWaiting(**Booking *booking***)**

## 2.2 Features to be tested

* ***High Risk***

1. **WaitingList class**

* To add users to the waiting list correctly based on their member type.
* To remove users who cancelled their booking from the waiting list.

1. **Booking class**

* To allocate room to the user following the room allocation rules.
* To fulfil the room quantity requested by the user.
* To test the redemption of exclusive rewards in room allocation for “member” users.
* To remove the bookings of the users from *waiting\_list* based on *member\_type*.
* To release the booked room from the system.

* ***Medium Risk***

1. **User class**

* To set the exclusive rewards after verifying the boolean value.

1. **Booking class**

* Retrieve the user from the booking and then return their user type.
* ***Low Risk***

1. **User class**

* To get user’s name.
* To get user’s member type.
* To retrieve the user's exclusive rewards and returns a Boolean value indicating whether the user has access to exclusive rewards or not.

1. **Booking class**

* To get the number of VIP rooms.
* To get the number of deluxe rooms.
* To get the number of standard rooms.

1. **WaitingList class**
   * To get the waiting list for each member types.

## 2.3 Features Not To Be Tested

1. **Printer class**

* To print the booking details of users.

1. **Room class**

* To check room availability based on room type.

1. **User class**

* To test the usage of exclusive reward for “VIP” users.

## 2.4 Test Basis

1. ***Requirement Documentation***

The project details show the requirements to be tested on the **Hotel Room Booking System**.

1. **Member type**

* There are three types of members, which are “VIP”, “member” or “normal”.
* The “VIP” or “member” users will be entitled to exclusive rewards occasionally.
* The exclusive reward of “VIP” users is not a feature to be tested.
* The exclusive reward of “member” users will be tested in the rules for room allocation.

1. **Default room type allocation based on member type**

* Default room type for “VIP” user is “VIP” room.
* Default room type for “member” user is “Deluxe” room.
* Default room type for "normal” user is “Standard” room.

1. **Rules for room allocation based on member type**
2. **“VIP” users**

* If the “VIP” room is fully booked, a “Deluxe” room will be priority allocated.
* If the “Deluxe” room is fully booked, a “Standard” room will be allocated.
* If the “Standard” room is fully booked, the users will be placed into “VIP” waiting list.

1. **“member” users**

* If the “Deluxe” room is fully booked, a “Standard” room will be allocated.
* If the “Standard” room is fully booked and the “VIP” room is available, the “VIP” room will only be allocated to the users with exclusive rewards and the exclusive reward will then be redeemed.
* If the “Standard” room is fully booked and the “VIP” room is not available, the users with or without exclusive reward will be placed into “member” waiting list.

1. **“normal” users**

* If the “Standard” room is fully booked, the users will be placed into “normal” waiting list.

1. **Maximum room quantity per booking based on member type**

* “VIP” users can book a maximum of three rooms at a time.
* “Member” users can book a maximum of two rooms at a time.
* “Normal” users can only book one room at a time.
* If the number of rooms requested cannot be fulfilled, the users will be placed into their respectively waiting lists based on their member type.

1. **Booking cancellation**

* If the user is on the waiting list, the user will be removed from the waiting list.
* If the user is not on the waiting list, the user will be removed from the booking list.

1. ***Design documentation***

The class diagram shows the five classes in the system under test, which are:

1. User
2. Booking
3. WaitingList
4. Room
5. Printer

Out of these five classes, User, Booking and WaitingList classes are ready for testing while Printer and Room classes are not ready for testing. Therefore, testing will focus on these three classes only. Moreover, the relationships between classes are shown with multiplicity:

1. **Association** relationship between User and WaitingList classes

* Multiplicity: one-to-many relationships

1. **Association** relationship between User and Booking classes

* Multiplicity: one-to many relationships

1. **Association** relationship between Booking and Room classes

* Multiplicity: one-to-many relationship.

## 2.5 Test Conditions

1. Verify that room numbers are updated accurately for all room types after every booking or cancellation.
2. Verify exclusive rewards are only awarded to “VIP” and “ember” users.
3. The room quantity requested for booking cannot be less than or equal to zero.

* **“VIP” users**

1. Verify that “VIP” user can book a maximum of three rooms at a time.
2. Verify that “VIP” user is allocated priority to the “VIP” rooms if available.
3. Verify that “VIP” user is then allocated priority to the “Deluxe” rooms when the “VIP” room is fully booked.
4. Verify that “VIP” user is allocated to the “Standard” rooms when both “VIP” and “Deluxe” rooms are fully booked.
5. Verify that “VIP” user is assigned the correct room combination based on availability.
6. Verify and confirm that “VIP” user is placed into the “VIP” waiting list when the requested number of rooms cannot be fulfilled.
7. Verify that “VIP” user can successfully cancel the booking by releasing the booked rooms.
8. Verify that “VIP” user in waiting list who cancel booking will be removed from the “VIP” waiting list.

* **“member” users**

1. Verify that “member” user can book a maximum of two rooms at a time.
2. Verify that “member” user is allocated priority to the “Deluxe” rooms if available.
3. Verify that “member” user is allocated to the “Standard” rooms when the “Deluxe” room is fully booked.
4. Confirm that whether “member” user has exclusive rewards.
5. Verify that when both “Deluxe” and “Standard” rooms are fully booked, the “VIP” rooms will be allocated to “member” users with the exclusive rewards only based on “VIP” rooms availability.
6. Verify that when “VIP”, “Deluxe” and “Standard” rooms are all fully booked, the “member” user with exclusive rewards will be placed into “Member” waiting list.
7. Verify that when both “Deluxe” and “Standard” rooms are fully booked, the “member” user without exclusive rewards will be placed into member” waiting list.
8. Verify that “member” user is assigned the correct room combination based on room availability.
9. Verify and confirm that “member” user is placed into the “member” waiting list when the requested number of rooms cannot be fulfilled.
10. Verify that “member” user can successfully cancel the booking and release the booked room
11. Verify that “member” user who is in waiting list can be removed from the “member” waiting list when user cancels booking.

* **“normal” users**

1. Verify that “normal” user can only book one room at a time.
2. Verify that “normal” user is allocated to the “Standard” rooms only if available.
3. Verify and confirm that “normal” user is placed into the “normal” waiting list when the requested number of rooms cannot be fulfilled.
4. Verify that “normal” user can successfully cancel the booking and release the booked room.
5. Verify that “normal” user who is in waiting list can be released from the “normal” waiting list when user cancels booking.
6. Verify that “normal” users cannot be assigned any exclusive rewards.

## 2.6 Entry Criteria

Entry Criteria are the conditions or prerequisites that must be satisfied before testing can begin. These standards ensure that the testing process is efficient and effective from the start.

1. Ensure all requirements related to hotel room booking system functionality and member type features are identified and documented.
2. Design decisions tables and test cases for booking creation, cancellation of booking, as well as add and remove waiting list to cover valid and invalid inputs.
3. The test cases should cover various scenarios including booking creation, cancellation, room availability checking and error handling.
4. Ensure all test cases, test data, test environment and any necessary tools are ready and available.
5. Prepare test data for simulating real-life scenarios of sample bookings during testing, including valid and invalid user inputs, different room type and scenarios where rooms are unavailable.
6. Ensure testing tools, frameworks and infrastructure such as Eclipse IDE, JUnit, JUnitParams and Mockito are set up and running.
7. Ensure the application code and functions ready for testing are completed.
8. Inform relevant team members of the start of the testing phase and their roles and responsibilities.

**2.7** **Exit Criteria**

Exit Criteria are the conditions that signify the completion of testing and enable the testing process to be formally concluded. These criteria ensure that testing has been thorough and successful before moving to the next phase of the project.

1. Verify that the system does not become inoperable due to failed updates during testing.
2. Ensure that the testing process achieves predetermined levels of test coverage, including functional coverage, code coverage, and requirements coverage.
3. All critical and high-priority defects discovered during testing must be fixed and verified as resolved. A predetermined percentage (95%) of medium and low priority defects should also be addressed.
4. Test metrics should show the stability and reliability of the system, completion of all planned tests, and adequate coverage of critical quality risks.
5. The system should show stability by running continuously for a predefined period (24 hours) without critical errors or crashes.
6. Ensure all necessary documentation, including test reports, are completed and reviewed for accuracy and completeness.
7. The team agrees to exit the testing phase in a Test Phase Exit Meeting.