X-Reserve

Software Testing

Version 1.0

**November 16, 2009**



**Pod 1**

|  |  |
| --- | --- |
| Sean Clark | 36538056 |
| Yang Gao | 52588050 |
| Shen Li | 65962060 |
| Neil Gentleman | 62973029 |
| Arash Malekzadeh | 33685058 |
| Michael Tando | 79529061 |
| Wei-Chen Wang | 64341043 |

Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 20-11-2009 | 1.0 | Document template |
| 26-11-2009 | 2.0 | Added Introduction |
| 26-11-2009 | 3.0 | Added functional requirements |
| 27-11-2009 | 4.0 | Added nonfunctional requirements |
| 28-11-2009 | 5.0 | Added overview and test schedule |
|  |  |  |
|  |  |  |

Table of Contents

1. Introduction 4

2. Overview 4

2.1 Scope and Objectives 4

2.2 Testing Criteria 4

2.2.1 Entry Criteria 5

2.2.2 Exit Criteria 5

2.3 Test Cycles and Schedule 6

2.4 Assumptions and Constraints 7

3. Functional Requirements Testing 7

3.1 Unit Testing 7

3.1.1 Black Box Testing 7

3.1.2 White Box Testing 7

3.2 Integration Testing 7

3.3 System Testing 8

3.3.1 Use Case 1 8

3.3.2 Use Case 2 9

3.3.3 Use Case 3 11

3.3.4 Use Case 4 12

3.3.5 Use Case 5 13

3.3.6 Use Case 6 15

3.3.7 Use Case 7 15

3.3.8 Use Case 8 17

3.3.9 Use Case 9 18

3.3.10 Use Case 10 19

3.3.11 Use Case 11 22

3.3.12 Use Case 12 23

3.3.13 Use Case 13 23

3.3.14 Use Case 14 24

3.3.15 Use Case 15 25

3.3.16 Use Case 16 25

3.3.17 Search Functions 26

3.3.18 Search Functions 28

4. Non-functional Requirements 28

4.1 Usability Testing 28

4.2 Scalability Testing 28

4.3 Performance Testing 29

4.4 Security Testing 30

5. Appendix A – Usability Test Results 32

6. Appendix B – Performance Test Results 32

7. Appendix C – Test Schedule 34

X-Reserve Software Testing

# 

# Introduction

X-Reserve is a web-based hotel reservation application that provides many hotel management features. This application can generate statistic reports to the hotel managers and allow them to manage user accounts. Hotel staffs are able to check-in customers and check-out customers. Moreover, X-Reserve provides a web-based interface for hotel guests to manage their accounts and reservations.

This test plan will be divided into three sections. The first section summaries the general environment of the application and the testing criteria of the application. This section will provide an overview of the X-Reserve system and list all the assumptions and constraints. Based on the environment of the system, all the testing criteria will be listed; also, the schedule of the test phase will be presented at the end of this section. The next two sections will present the detailed testing procedures for functional requirements and non-functional requirements. All the testing procedures will be executed many times at different development phases to ensure the accuracy and quality of the system.

# Overview

## Scope and Objectives

The main objective of this software test plan is to verify and validate our X-reserve hotel management software system in order to ensure that the software meets the business and technical requirements that guided its design and development and works as expected. As part of the objective, this test plan aims to discover as many undiscovered errors as possible through a rigorous set of carefully chosen test cases. Some aspects of software quality can be measured by thorough testing; however, the quality assurance aspect of software development is out of scope of this test plan. This test plan aims to measure the quality of the software, not to improve it.

This test plan is broken down into sections that evaluate the functional requirements at unit, integrated, and system levels, as well as the non-functional requirements such as performance, usability, reliability, security, and adaptability.

## Testing Criteria

This section contains a list of criteria that the software application must satisfy. The testing criteria are divided into 2 sub-sections: entry criteria and exit criteria. Entry criteria denote the conditions that must be present before testing process can begin. Exit criteria denote the conditions or process that must be present before a test cycle completes.

### Entry Criteria

Usability Testing

* Verify if a set of test participants find the application interface understandable and contains no ambiguity.
* Verify if a set of test participants find the application installation and execution easy.
* Verify if a set of test participants find the application useful and interesting.
* Verify the satisfactory level of a set of test participants using the application.

Scalability Testing

* Verify if the application database can store up to 1000 user accounts and 100 rooms.
* Verify if the application can withstand 20 simultaneous user requests.

Performance Testing

* Verify the expected application response time of less than 1 second for loading application web page using a typical connection speed of 128kbps.
* Verify the expected application response time with more than 5 simultaneous users running the application.

Use Case Function Testing

* Verify the application response to wrong inputs e.g. entering negative numbers, null fields, boundary conditions, error-handling paths.
* Verify if the application writes/fetches correct data to/from database.

Role-Based Access Control Testing

* Verify if each role has the correct privileges assigned to them according to the requirements.

### Exit Criteria

* All the test cases have been executed.
* All high priority errors must be fixed and tested.
* Application must provide all the required services according to the requirements specification document.

## Test Cycles and Schedule

This section provides an outline of our software development test cycle. The testing process is broken down into 4 sections: Unit testing, integration testing, system testing, and validation. Unit testing is conducted using black box and white box testing techniques to verify the internal logic and data structures of each unit. Once the individual modules have been tested, we can proceed to integration testing. The integration testing predominantly exercises black box testing to verify that the modules communicate and interact correctly with other modules in the system. The next step in the testing cycle is the manual system testing which involves thorough walkthrough of every use case functions. The final step is to validate the system, ensuring that we have built the right product before releasing it. The diagram below illustrates the software test cycle. The details of testing processes are discussed further in the later section. The schedule of the tests is included in Appendix C.

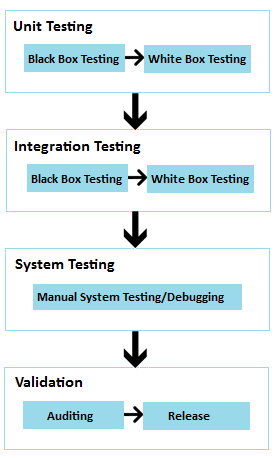


Figure 1. Test Cycle

## Assumptions and Constraints

Security testing covers a lot of aspects of the system such as data integrity, confidentiality, authentication, authorization, etc. Since many of the security issues are handled by the Spring framework, this test plan will assume those security issues are handled properly by Spring.

# Functional Requirements Testing

## Unit Testing

The goal of unit testing is to divide a piece of source code to various parts and examined each part’s behavior by isolating them from remainder. The unit testing is accomplished by the combination of black box and white box testing while adopting the JUnit testing, a unit testing framework for Java. This adoption assists the creation and execution of test cases.

### Black Box Testing

Black box testing is a testing method that treats the software as if it has no knowledge of the internal implementation while testing. This testing aims to test how well the functionality of the software conforms to the targeted requirements. Thus, testers inputs data and examine the output from the tested object. Then they can distinguish whether the behavior of the tested object has the same expected outcome. The input specifications include boundary conditions, exceptions, nullity, invalidation, and aliasing problems. The advantage of this test is that it can uncover the conceptual errors that debuggers are unable to find.

### White Box Testing

In contrast, white box testing is a testing method that focuses specifically on using internal knowledge of the software. Our test strategies incorporate coverage of the followings:

* Static analysis – examines codes and determines flaws
* Dynamic analysis – executes methods and analyzes outputs
* Failure analysis – executes all possible failure cases to cause the program crashes
* Statement coverage – executes all statements and makes sure at least 1 successful trial
* Branch coverage – executes all branches and makes sure no abnormal behaviors
* Loop coverage – executes all the loops are covered 0, 1, 2 times
* Path coverage – executes all the path and checks for consistency in behaviors

## Integration Testing

The goal of integration testing is to examine all of the features are developed and work together corporately. In other words, is the product good enough to be delivered? First, integration testing will be performed whenever a module is completed and has passed the unit and functional testing phase. Then testers will combine this module with the others to determine if the aggregates function correctly together. The sequence of the testing is from the lowest level of component (smallest aggregates) to the highest level of component (largest aggregates). This is also known as the bottom-up integration testing.

## System Testing

### Use Case 1

**Test Case:** Create An Account (invalid input)

**Actor:** Guest

**Pre-conditions:** true

**Detailed Description:** This test case examines the functionality of creating accounts with invalid input

**Test Procedure:**

1.0 Selects "Register" to apply for an account

2.0 Enters invalid user name or password in the corresponding fields

2.1 Enters only one of the fields

2.2 Enters none of the fields

3.0 Selects "Register" to submit the application

**Expected Results:** The system will indicate username and password fields cannot be empty

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | The system indicates username and password fields cannot be null. | Pass |
| Oct. 24, 2009 | Same result as above | Pass |
| Nov. 1, 2009 | Same result as above | Pass |
| Nov. 14, 2009 | Same result as above | Pass |
| Nov. 26, 2009 | Same result as above | Pass |

**Test Case:** Create An Account (duplicate account)  
**Actor:** Guest  
**Pre-conditions:** An account with user name "user" exists  
**Detailed Description:** This test case examines the functionality of creating duplicate accounts  
**Test Procedure:**  
  1.0 Selects "Register" to apply for an account  
    2.0 Enters "user" in the user name field

     3.0 Enters a password in the password field

    4.0 Selects "Register" to submit the application  
**Expected Results:** The system will indicate username has already been used  
**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates the another entity has the same value | Pass |
| Oct. 24, 2009 | Same result as above | Pass |
| Nov. 1, 2009 | Same result as above | Pass |
| Nov. 14, 2009 | Same result as above | Pass |
| Nov. 26, 2009 | Same result as above | Pass |

**Test Case:** Create An Account (valid input)  
**Actor:** Guest  
**Pre-conditions:** Valid personal information (name & password)  
**Detailed Description:** This test case examines the functionality of creating accounts with valid input    
**Test Procedure:**  
  1.0 Selects "Register" to apply for an account  
    2.0 Enters valid user name or password in the corresponding fields

    3.0 Selects "Register" to submit the application  
**Expected Results:** The system will automatically login guest to search page

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The new account has been successfully created and the guest is automatically logged in | Pass |
| Oct. 24, 2009 | Same result as above | Pass |
| Nov. 1, 2009 | Same result as above | Pass |
| Nov. 14, 2009 | Same result as above | Pass |
| Nov. 26, 2009 | Same result as above | Pass |

### Use Case 2

**Test Case:** Make a Reservation (valid input, valid payment and valid login)  
**Actor:** Guest

**Pre-conditions:** Guest logins to the system and searches for a set of rooms  
**Detailed Description:** This test case tests the functionality of making a reservation with valid input, valid payment and valid login  
**Test Procedure:**

    1.0 Selects a room type and clicks the "Reserve" link

    2.0 Enters check-in date, check-out date and number of people in the corresponding fields

    3.0 Clicks the "continue reservation" button

    4.0 After viewing the summary of the room selected and the information entered, selects a payment type from the drop down list (Visa or Master)

    5.0 Enters credit card information (credit card number, expired date, and name on the credit card)

    6.0 Clicks the confirm button  
**Expected Results:** The system will indicate that the reservation process is completed  
**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | The system indicates that the reservation process is completed | Pass |
| Oct. 24, 2009 | The system generates an error message | Fail |
| Nov. 1, 2009 | The system indicates that the reservation process is completed | Pass |
| Nov. 14, 2009 | Same result as above | Pass |
| Nov. 26, 2009 | The system generates an error message | Fail |
|  |  |  |

**Test Case:** Make a Reservation (invalid input)  
**Actor:** Guest

**Pre-conditions:** Guest logins to the system and searches for a set of rooms  
**Detailed Description:** This test case tests the functionality of making a reservation with invalid input  
**Test Procedure:**

    1.0 Selects a room and clicks the "continue reservation" button

    2.0 Enters invalid check-in date or invalid check-out date or invalid number of people in the corresponding fields

        2.1 Enters a check-in date that is before today

        2.2 Enters a check-out date that is before check-in date

        2.3 Enters a check-out date that is before today

        2.4 Enters information only in two of the fields

        2.5 Enters information only in one of the fields

        2.6 Inserts empty field

    3.0 Clicks the "continue reservation" button

**Expected Results:** The system will indicate that one or more input are invalid or missing

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system does not detect invalid or missing input | Fail |
| Oct. 24, 2009 | Same result as above | Fail |
| Nov. 1, 2009 | Same result as above | Fail |
|  |  |  |
|  |  |  |

**Test Case:** Make a Reservation (valid input, valid login, invalid payment)  
**Actor:** Guest  
**Pre-conditions:** Guest logins to the system and searches for a set of rooms  
**Detailed Description:** This test case tests the functionality of making a reservation with valid input, valid login but invalid payment input  
**Test Procedure:**

    1.0 Selects a room and clicks the "continue reservation" button

    2.0 Enters check-in date, check-out date and number of people in the corresponding fields

    3.0 Clicks the "continue reservation" button

    4.0 After viewing the summary of the room selected and the information entered, selects a payment type from the drop down list (Visa or Master)

    5.0 Enters invalid credit card information (credit card number, expired date, and name on the credit card)

    5.1 Enters information only in two of the fields

    5.2 Enters information only in one of the fields

    5.3 Inserts empty field

    5.4 Enters a credit card number that is less than 16 digits

    5.5 Enters an expired date that is before today

    6.0 Clicks the confirm button

**Expected Results:** The system will indicate that one or more input are invalid or missing

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system does not detect invalid or missing input | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 3

**Test Case:** Guest Cancels a Reservation  
**Actor:** Guest  
**Pre-conditions:** Guest logins to the system  
**Detailed Description:**  This test case examines the functionality of cancelling a reservation by a guest  
**Test Procedure:**

    1.0 Clicks the "view reservation(s)" button

    2.0 Selects a reservation and click the "cancel" button

    3.0 After viewing a summary of the reservation and the cancellation fee, clicks the "confirm" button

**Expected Results:** The reservation is cancelled and the fee can be viewed inside the "view bill" option

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not Yet Implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Staff Cancels a Reservation  
**Actor:** Staff  
**Pre-conditions:** Staff logins to the system, requires guest information (guest name, guest address) and searches for guest information  
**Detailed Description:**  This test case examines the functionality of cancelling a reservation by a staff  
**Test Procedure:**

    1.0 Clicks the "view reservation(s)" button

    2.0 Selects a reservation and click the "cancel" button

    3.0 After viewing a summary of the reservation and the cancellation fee, click the "confirm" button

**Expected Results:** The reservation is cancelled and the fee can be viewed inside the "view bill" option of the guest

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not Yet Implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 4

**Test Case:** Create Rooms (valid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of creating rooms with valid input  
**Test Procedure:**  
    1.0 Selects "create room"

    2.0 Selects a room type from the "Room Type" drop down list

    3.0 Enters room number in the "Room Number" field  
    4.0 Selects "Save"  
**Expected Results:** The system will confirm the operation and the new room can be viewed inside the "view room" option

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully creates the new room | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Create Rooms (duplicate room number)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system and a room with room number "110" exists  
**Detailed Description:** This test case examines the functionality of creating duplicate rooms    
**Test Procedure:**  
    1.0 Selects "create room"

    2.0 Selects a room type from the "Room Type" drop down list

    3.0 Enters "110" in the "Room Number" field  
    4.0 Selects "Save"

**Expected Results:** The system will indicate the room number has been used

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates that the room number has been used | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Create Rooms (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of creating rooms with invalid input  
**Test Procedure:**  
    1.0 Selects "create room"

    2.0 Selects a room type from the "Room Type" drop down list

    3.0 Enters invalid input in the "Room Number" field

        3.1 Enters a negative number in the "Room Number" field

        3.2 Enters nothing in the "Room Number" field

    4.0 Selects "Save"

**Expected Results:**

        3.1 The system will display an error message saying the room number cannot be negative

        3.2 The system will display an error message saying the room number field can't be empty  
**Test Results:**

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | 3.1 The system does not display error message and created new room with invalid room number  3.2 The system displays an error message | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 5

**Test Case:** Edit Rooms (valid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of editing rooms with valid input  
**Test Procedure:**   
    1.0 Clicks the "view rooms" button  
    2.0 Chooses a room and selects "edit"

    3.0 Selects a room type from the "Room Type" drop down list

    4.0 Enters a new room number

    5.0 Selects "Save"  
**Expected Results:** The system will update the changes to the room

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully updates the changes to the room | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Edit Rooms (duplicate room number)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system and room number "110" and "111" exist  
**Detailed Description:** This test case examines the functionality of editing rooms with duplicate room number  
**Test Procedure:**   
    1.0 Clicks the "view rooms" button  
    2.0 Chooses room "110" and selects "edit"

    3.0 Selects a room type from the "Room Type" drop down list

    4.0 Enters "111" in the "Room Number" field

    5.0 Selects "Save"  
**Expected Results:** The system will indicate room "111" is already exists

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates room "111" is already exists | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Edit Rooms (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of editing rooms with invalid input  
**Test Procedure:**   
    1.0 Clicks the "view rooms" button  
    2.0 Chooses a room and selects "edit"

    3.0 Select a room type from the "Room Type" drop down list

    4.0 Enters invalid input in the "Room Number" field

        4.1 Enters a negative number in the "Room Number" field

        4.2 Enters nothing in the "Room Number" field

    5.0 Selects "Save"  
**Expected Results:**  
        4.1 The system will display an error message saying the room number cannot be negative

        4.2 The system will display an error message saying the room number field can't be empty

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | 4.1 The system does not display error message and created new room with invalid room number 4.2 The system displays an error message | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 6

**Test Case:** Delete Rooms  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system and there must be at least a room to delete  
**Detailed Description:** This test case examines the functionality of deleting rooms  
**Test Procedure:**   
    1.0 Clicks the "view rooms" button

    2.0 Chooses a room and selects "delete"  
**Expected Results:** The system indicates that the selected room has been deleted  
**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully deletes the selected room | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 7

**Test Case:** Create Room Type (valid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of creating room type with valid inputs  
**Test Procedure:**   
    1.0 Selects "Create Room Type"

    2.0 Enters valid type name, description, daily rate and maximum occupancy in the corresponding fields  
    3.0 Select "save"  
**Expected Results:** The system will add a new room type

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully created the new room type | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Create Room Type (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of creating room type with invalid inputs  
**Test Procedure:**   
    1.0 Selects "Create Room Type"

    2.0 Enters invalid inputs in the corresponding fields (type name, description, daily rate and maximum occupancy)

        2.1 Enters zero of the fields

        2.2 Enters only one of the fields

        2.3 Enters only two of the fields

        2.4 Enters only three of the fields

        2.5 Enters a negative number in the daily rate field

        2.6 Enters a string of characters in the daily rate field

        2.7 Enters a decimal number in the maximum occupancy field

        2.8 Enters a negative number in the maximum occupancy field

        2.9 Enters a string of characters in the maximum occupancy field  
    3.0 Select "save"  
**Expected Results:**

        2.1, 2.2, 2.3, 2.4 The system will indicate that one or more fields are missing

        2.5, 2.6, 2.7, 2.8, 2.9 The system will indicate that one ore more inputs are invalid

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | 2.1, 2.2, 2.3, 2.4  The system indicates that one or more fields are missing 2.5 The system does not detect the invalid input 2.6 The system displays an error message 2.7 The system displays an error message 2.8 The system does not detect the invalid input 2.9 The system displays an error message | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Create Room Type (duplicate room type)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system and a room type called "VIP" exists  
**Detailed Description:** This test case examines the functionality of creating duplicate room type  
**Test Procedure:**   
    1.0 Selects "Create Room Type"

    2.0 Enters "VIP" in the type name field

    3.0 Enters a description, daily rate and maximum occupancy in the corresponding fields

    4.0 Select "save"  
**Expected Results:** The system will indicate that the room type is already exists

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates that the room type is already exists (Case Insensitive ) | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 8

**Test Case:** Edit Room Type (valid input)  
**Actor:** Admin  
**Pre-conditions:** Admin needs to log into admin account and there must be at least a room type to edit  
**Detailed Description:**  This test case examines the functionality of editing room type with valid input  
**Test Procedure:**  
    1.0 Selects "View Rooms Type"  
    2.0 Chooses a room type and selects "edit"

    3.0 Enters room type name, description, daily rate and Max occupancy in the corresponding fields  
    4.0 Select "save"  
**Expected Results:** The system will indicate that the operation is complete and the room type has been modified

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully saves the changes made to the room type | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Edit Room Type (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin needs to log into admin account and there must be at least a room type to edit  
**Detailed Description:**  This test case examines the functionality of editing room type with invalid input  
**Test Procedure:**  
    1.0 Selects "View Rooms Type"  
    2.0 Chooses a room type and selects "edit"  
    3.0 Enters invalid room type name, description, daily rate or Max occupancy in the input fields

        3.1 Enters zero of the fields

        3.2 Enters only one of the fields

        3.3 Enters only two of the fields

        3.4 Enters only three of the fields

        3.5 Enters a negative number in the daily rate field

        3.6 Enters a negative number in the Max occupancy field

        3.7 Enters zero in the daily rate field

        3.8 Enters zero in the Max occupancy field

        3.9 Enters a string of characters in the daily rate field

        3.10 Enters a string of characters in the Max occupancy field

    4.0 Select "save"  
**Expected Results:**

        3.1, 3.2, 3.3, 3.4 The system will indicate that one or more fields are missing

        3.5, 3.6, 3.7, 3.8, 3.9, 3.10 The system will indicate that some inputs are invalid  
**Test Results:**  
        3.1, 3.2, 3.3, 3.4 The system will indicate that one or more fields are missing  
        3.5 The system does not detect the invalid input  
        3.6 The system does not detect the invalid input  
        3.7 The system does not detect the invalid input  
        3.8 The system does not detect the invalid input  
        3.9 The system displays an error message  
        3.10 The system displays an error message

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | 3.1, 3.2, 3.3, 3.4 The system will indicate that one or more fields are missing 3.5 The system does not detect the invalid input 3.6 The system does not detect the invalid input  3.7 The system does not detect the invalid input 3.8 The system does not detect the invalid input 3.9 The system displays an error message 3.10 The system displays an error message | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 9

**Test Case:** Delete Room Type  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system and there must be at least a room type to delete  
**Detailed Description:**  This test case examines the functionality of deleting room type  
**Test Procedure:**   
    1.0 Selects "View Room Types"  
    2.0 Chooses a room type and selects "delete"  
**Expected Results:** The system will indicate the selected room type is deleted  
**Test Results:** The system successfully deletes the selected room type

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates username and password fields cannot be null. | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 10

**Test Case:** Add Chargeable Items (valid input)   
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of adding chargeable items with valid input  
**Test Procedure:**   
    1.0 Selects "Create Chargeable Item"

    2.0 Enters valid item name, description, and price in the corresponding field

    3.0 Selects "Save"  
**Expected Results:** The system will indicate the new chargeable item is created

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully creates the new chargeable item | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Add Chargeable Items (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of adding chargeable items with invalid input  
**Test Procedure:**   
    1.0 Selects "Create Chargeable Item"

    2.0 Enters invalid item name, description, and price in the corresponding field

        2.1 Enters zero of the fields

        2.2 Enters only one of the fields

        2.3 Enters only two of the fields

        2.4 Enters a negative number in the price field

        2.5 Enters a string in the price field

    3.0 Selects "Save"  
**Expected Results:**  
    2.1 The system will display an error message saying one or more field is missing  
    2.2 The system will display an error message saying one or more field is missing  
    2.3 The system will display an error message saying one or more field is missing

    2.4 The system will indicate the input is invalid  
    2.5 The system will indicate the input is invalid

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | 2.1 The system does not detect the invalid input  2.2 The system does not detect the invalid input  2.3 The system does not detect the invalid input  2.4 The system does not detect the invalid input 2.5 The system displays an error message | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Add Chargeable Items (duplicate item)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system and a chargeable item named "Item1" exists  
**Detailed Description:** This test case examines the functionality of adding duplicate chargeable items  
**Test Procedure:**   
    1.0 Selects "Create Chargeable Item"

    2.0 Enters "Item1" in the item name field

    3.0 Enters a description and price in the corresponding field

    4.0 Selects "Save"  
**Expected Results:** The system will indicate that "Item1" has already existed

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system does not detect the duplicate chargeable item name (detects SKU duplicity ) | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Edit Chargeable Items (valid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of editing chargeable items with valid input  
**Test Procedure:**   
    1.0 Selects "View Chargeable Item"  
    2.0 Chooses an item and selects "edit"

    3.0 Enters a new valid item name, description, and price in the corresponding field

    4.0 Selects "complete"  
**Expected Results:** The system will indicate the operation is completed

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system successfully saves the changes made to the selected chargeable item | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Edit Chargeable Items (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of editing chargeable items with invalid input  
**Test Procedure:**   
    1.0 Selects "View Chargeable Item"  
    2.0 Chooses an item and selects "edit"

    3.0 Enters invalid input in the price field

        3.1 Enters a negative number in the price field

    4.0 Selects "complete"  
**Expected Results:** The system will indicate the price cannot be zero  
**Test Results:** The system does not detect the invalid input

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates username and password fields cannot be null. | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Delete Chargeable Items   
**Actor:** Admin  
**Pre-conditions:** Admin needs to log into admin account  
**Detailed Description:** This test case examines the functionality of deleting chargeable items   
**Test Procedure:**   
    1.0 Selects "View Chargeable Item"  
    2.0 Chooses an item and selects "delete"  
**Expected Results:** The system will indicate the selected chargeable item is deleted  
**Test Results:** The system successfully deletes the selected chargeable item

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates username and password fields cannot be null. | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 11

**Test Case:** Charge Chargeable Items (valid input)  
**Actor:** Staff  
**Pre-conditions:** Staff logins to the system and searches for a customer  
**Detailed Description:** This test case examines the functionality of charging chargeable items with valid input  
**Test Procedure:**

    1.0 Clicks the "Add Chargeable Item" button

    2.0 Select a chargeable item from the drop down list

    3.0 Enters a positive quantity in the "Quantity" field

    4.0 After viewing the total amount of the chargeable items, clicks the "complete" button  
**Expected Results:** The system indicate the operation is completed and the modified bill can be viewed in the "view bill" option

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Charge Chargeable Items (invalid input)  
**Actor:** Staff  
**Pre-conditions:** Staff must be logged in to the system  
**Detailed Description:** This test case examines the functionality of charging chargeable items with invalid input  
**Test Procedure:**

    1.0 Clicks the "Add Chargeable Item" button

    2.0 Select a chargeable item from the drop down list

    3.0 Enters invalid input in the "Quantity" field

        3.1 Enters a negative number

        3.2 Enters nothing

    4.0 After viewing the total amount of the chargeable items, clicks the "complete" button  
**Expected Results:**

        3.1 The system will indicate that the quantity cannot be negative

        3.2 The system will indicate that the quantity field cannot be empty

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 12

**Test Case:** Check-in  
**Actor:** Staff  
**Pre-conditions:** Staff must be logged in to the system  
**Detailed Description:** This test case examines the functionality of checking in  
**Test Procedure:**

    1.0 clicks the "check-in" button

    2.0 enters customer name, customer phone number, customer address

    3.0 clicks the "search" button

    4.0 after viewing a list of reservations by that customer, select one reservation and clicks "continue"

    5.0 after viewing a summary of the reservation, clicks the "complete" button  
**Expected Results:** The system will indicate that the check-in process is completed, and the reservation in the customer's account will be gone

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 13

**Test Case:** Check-out (pay by cash)  
**Actor:** Staff  
**Pre-conditions:** Staff must be logged in to the system  
**Detailed Description:** This test case examines the functionality of checking out  
**Test Procedure:**

    1.0 clicks the "check-out" button

    2.0 enters customer name, customer phone number and customer address

    3.0 clicks the "search" button

    4.0 after viewing a list of transactions by that customer, select one transaction and clicks "continue"

    5.0 after viewing a summary of the transaction, clicks the "continue" button

    6.0 selects "cash" as the payment type

    7.0 enters the amount of cash the customer gives and clicks the "continue" button

    7.0 after viewing the changes needed to give the customer, clicks the "complete" button  
**Expected Results:** The system will indicate that the check-out process is completed

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Check-out (pay by credit card)  
**Actor:** Staff  
**Pre-conditions:** Staff must be logged in to the system  
**Detailed Description:** This test case examines the functionality of checking out  
**Test Procedure:**

    1.0 clicks the "check-out" button

    2.0 enters customer name, customer phone number and customer address

    3.0 clicks the "search" button

    4.0 after viewing a list of transactions by that customer, select one transaction and clicks "continue"

    5.0 after viewing a summary of the transaction, clicks the "continue" button

    6.0 selects "credit card" as the payment type

    7.0 clicks the "complete" button  
**Expected Results:** The system will indicate that the check-out process is completed

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 14

**Test Case:** Check Dirty Rooms   
**Actor:** Staff  
**Pre-conditions:** Staff logins to the system   
**Detailed Description:** This test case examines the functionality of checking dirty rooms  
**Test Procedure:**

    1.0 clicks the "view dirty rooms" button

**Expected Results:** The system will display a set of rooms that is dirty

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 15

**Test Case:** Mark Clean Rooms   
**Actor:** Staff  
**Pre-conditions:** Staff logins to the system  
**Detailed Description:** This test case examines the functionality of marking clean rooms   
**Test Procedure:**

    1.0 clicks the "view dirty rooms" button

    2.0 selects a room

    3.0 clicks the "mark it as clean" button

    4.0 after viewing the summary of that room, clicks the "confirm" button  
**Expected Results:** the system will display a confirm message - "the selected room has been marked as clean"

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Use Case 16

**Test Case:** View Statistic Report (valid input)   
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of viewing statistic report with valid input  
**Test Procedure:**

    1.0 Clicks the "view report" button

    2.0 Selects a report type from the "Report Type" drop down list

    3.0 Enters a starting date and an ending date

    4.0 Clicks the "View" button  
**Expected Results:** The system will display the report calculated based on the period provided

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** View Statistic Report (invalid input)  
**Actor:** Admin  
**Pre-conditions:** Admin must be logged in to the system  
**Detailed Description:** This test case examines the functionality of viewing statistic report with invalid input  
**Test Procedure:**

    1.0 Clicks the "view report" button

    2.0 Selects a report type from the "Report Type" drop down list

    3.0 Enters a invalid starting date or a invalid ending date

        3.1 Enters an ending date that is before the starting date

    4.0 Clicks the "View" button  
**Expected Results:** The system will indicate the period provided is undefined  
**Test Results:** ( Not yet implemented )

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | The system indicates username and password fields cannot be null. | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Search Functions

**Test Case:** Search for a set of rooms  
**Actor:** Guest or Staff or Admin  
**Pre-conditions:** true  
**Detailed Description:**  This test case examines the functionality of searching for a set of rooms  
**Test Procedure:**  
    1.0 Selects "Home" to be linked to search function  
    2.0 Specifies a price range

    3.0 Enters number of entries

    4.0 Enters number of beds required

    5.0 Selects a room type from the drop down list

    6.0 Enters check-in date and check-out date

    7.0 Selects "Search" to search for the rooms

**Expected Results:** The system will display a set of rooms according to the specification

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Staff searches for guest information (valid input)

**Actor:** Staff  
**Pre-conditions:** Staff logins to the system  
**Detailed Description:**  This test case examines the functionality of retrieving a guest's information by a staff  
**Test Procedure:**

    1.0 Enters guest name and guest address in the search bar

    2.0 Clicks the "search" button

    3.0 Clicks the "view account" button  
**Expected Results:** The system will display the guest's account information

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case:** Staff searches for guest information (invalid input)

**Actor:** Staff  
**Pre-conditions:** Staff logins to the system  
**Detailed Description:**  This test case examines the functionality of retrieving a guest's information by a staff  
**Test Procedure:**

    1.0 Enters invalid guest information (guest name and guest address) in the search bar

        1.1 Enters only one of the fields

        1.2 Leaves all the fields empty

        1.3 Enters an incorrect guest name

        1.4 Enters an incorrect guest address

        1.5 Enters incorrect guest name and incorrect guest address

    2.0 Clicks the "search" button

**Expected Results:** The system will indicate that "no user is found" or "one or more fields are missing"

**Test Results:**

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct 2, 2009 | Not yet implemented | N/A |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Search Functions

# Non-functional Requirements

## Usability Testing

Usability testing focuses on the simplicity of the system. Users tend to use systems that are easy to use or they can learn how to use in a very short time. As a result, usability testing is very important. We will use the hallway testing to verify the usability requirements of the X-Reserve software.

**Test Method**: Hallway Testing

**Actor**: five to six test participants

**Pre-condition**: all test participants must never use our software before and they must not be familiar with this kind of web-based hotel management system.

Detailed Description: Each test participant must perform the following tasks on their own. After performing all the tasks, they should rate their satisfaction and provide some feedbacks if they have any.

**Test Tasks**:

1. Creates an account
2. Login to the system
3. Make a reservation
4. Cancel a reservation
5. Search for a room type

**Expected Results**: Satisfaction Level must be greater or equal than four for all tasks

**Test Results**: The test results are listed in Appendix A

## Scalability Testing

Scalability refers to the acceptance of increase in volume of customer data and business data. As for the requirements of the X-Reserve system, it should be able to store at least 1000 customers and at least 100 rooms. Also, our system should be able to handle over twenty simultaneous requests at the same time.

**Test Case**: Create 1000 user accounts

**Actor**: Tester

**Pre-condition**: Creates a test plan that automatically create 1000 user accounts using jMeter

**Detailed Description**: This test case examines the scalability of increased customer volume

**Test Procedure**: Execute the test plan

**Expected Outcome**: All create account operations are completed

**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | 1000 user accounts have been created | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case**: Create 100 rooms

**Actor**: Tester

**Pre-condition**: Creates a test plan that automatically create 100 rooms using jMeter

**Detailed Description**: This test case examines the scalability of increased customer volume

**Test Procedure**: Execute the test plan

**Expected Outcome**: All create account operations are completed

**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | 1000 user accounts have been created | Pass |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Test Case: Create 100 rooms

Test Case: handle twenty simultaneous requests

## Performance Testing

Performance testing is used to test the speed of the software. The purpose of this test is to verify that our software satisfies the performance requirements. Performance usually can refer to many different aspects of a system. But in this project, performance will refer to the response time of the application. The following test cases will be used to execute performance testing.

**Test Case**: Measure the speed of requesting all the pages

**Test Tool**: JMeter

**Actor**: Tester

**Pre-condition:** Using JMeter to create a test plan with one user and all the HTTP requests to all the pages.

**Detailed Description**: The speed of retrieving all the web pages will be measured by using JMeter.

**Test Procedure**: execute the test plan

**Expected Outcome**: A summary table will be displayed. Since performance varies depending on the network speed. As a result, loading a page within 1 second will be our expected outcome. A typical page has a size of 25 Kilobytes, so our expected speed will be 25 KB/second.

**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Nov. 25, 2009 | The results are listed in Appendix B | Pass |

**Test Case**: Measure the speed of 5 users requesting all the pages for 100 times at the same time

**Test Tool**: JMeter

**Actor**: Tester

**Pre-condition:** Using JMeter to create a test plan with 5 users, 100 loop times and all the HTTP requests to all the pages.

**Detailed Description**: The speed of retrieving all the web pages will be measured by using JMeter.

**Test Procedure**: execute the test plan

**Expected Outcome**: A summary table will be displayed. Since performance varies depending on the network speed. As a result, loading a page within 1 second will be our expected outcome. A typical page has a size of 25 Kilobytes, so our expected speed will be 25 KB/second.

**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Nov. 25, 2009 | The results are listed in Appendix B | Pass |

## Security Testing

Security testing covers a lot of aspects of the system such as data protection. Since many of the security issues are handled by the Spring framework, we will assume those security issues are handled properly by Spring. This test plan will only cover the role hierarchy security requirements. The following test case illustrates the test on role hierarchy security.

**Test Case**: Admin deletes a user account

**Actor**: Admin

**Pre-condition:** Admin logins to his/her account and a user has username as “user” existed in the system

**Detailed Description**: This test case examines the role hierarchy of the system

**Test Procedure**:

1.0 Clicks on the “User” link on the top menu

2.0 Move the mouse upon the user called “user”

3.0 Clicks the “delete” link on the right hand side

**Expected Outcome**: The user with user name “user” will be deleted

**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | An error page is displayed | Fail |
| Oct. 24, 2009 | An error page is displayed | Fail |
| Nov. 1, 2009 | The specified user is deleted | Pass |
| Nov. 14, 2009 | Same result as above | Pass |
| Nov. 26, 2009 |  |  |

**Test Case**: User violates the role hierarchy

**Actor**: User

**Pre-condition:** User logins to his/her account

**Detailed Description**: This test case examines the role hierarchy of the system

**Test Procedure**:

1.0 User should not have any privilege to do any of the following

1.1 User cannot view all user accounts

1.2 User cannot delete Staff accounts

1.3 User cannot modify Staff accounts

1.4 User cannot delete Admin accounts

1.5 User cannot modify Admin accounts

1.6 User cannot view all reservations

**Expected Outcome**: The user with user name “user” will be deleted

**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | 1.1 and 1.6 are failed. User is able to view all reservations and all users | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Test Case**: Staff violates the role hierarchy

**Actor**: Staff

**Pre-condition:** Staff logins to his/her account

**Detailed Description**: This test case examines the role hierarchy of the system

**Test Procedure**:

1.0 Staff should not have any privilege to do any of the following

1.1 Staff cannot view all user accounts

1.2 Staff cannot delete user accounts

1.3 Staff cannot modify user accounts

1.4 Staff cannot delete Admin accounts

1.5 Staff cannot modify Admin accounts

**Expected Outcome**: The user with user name “user” will be deleted

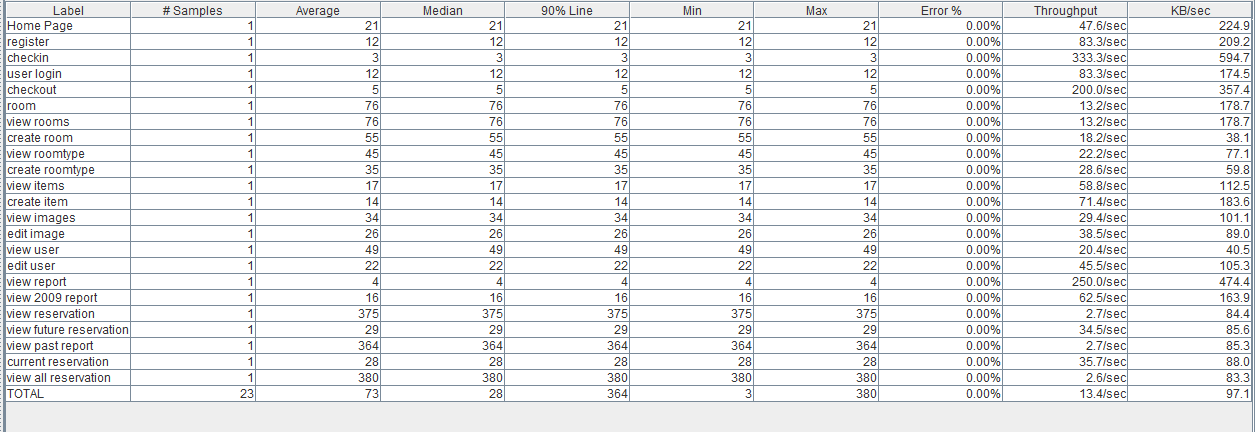
**Test Results**:

|  |  |  |
| --- | --- | --- |
| Date | Result | Pass/Fail |
| Oct. 2, 2009 | 1.1 is failed. Staff can view all users | Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Appendix A – Usability Test Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tester ID | Test Task | Satisfaction Level(out of 5) | Feedback | Test Date |
| 1 | Creates an account |  |  |  |
| 1 | Login to the system |  |  |  |
| 1 | Make a reservation |  |  |  |
| 1 | Cancel a reservation |  |  |  |
| 1 | Search for a room type |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Appendix B – Performance Test Results



# Appendix C – Test Schedule

