# Phase 4: Testing and Refactoring

1. **Introduction**

## *Purpose*

This document was written to demonstrate the testing & refactoring procedures conducted by the Chaotic Coders group, otherwise known as Group #3, in accordance with their hotel reservation application project. This application is designed to display hotel rooms from different hotels and allow users to create, edit, and cancel their reservations. In addition, it is designed to allow manager accounts for hotel moderation, with acts such as editing or adding rooms, or changing the details of a hotel. The program will also come with numerous other features, however this document will not go into the specifics of that but rather the process of how the code was refactored and tested to ensure proper efficiency. It was modified numerous times across development phases in order to exemplify an organized structure. The general look of the code has indeed changed thoroughly over time.

# Each member contributed to this project phase, and here are the contributions in detail:

**Alex**: This phase I was able to get the database fully integrated with the application and start only using the database instead of using a lot of memory on objects.

**Ethan**: In this phase of the project, I mainly did refactoring work and modified reservations a lot, so they now hold IDs instead of the actual objects. In addition, reservation methods are now obtained from the managers rather than the actual classes to improve organization and readability. Although a couple things still need to be modified, I was able to accomplish quite a few aspects of the refactoring for reservations.

**Minas**: In this phase of the project, I finished “Edit my account” with its different options such as change name, password, delete account etc. I started working on “View all reservations” for the manager, which will allow us to see all current reservations. Also started working on “Edit Hotel”.

**Nura**: In this phase of the project, I completely finished the Login and Sign Up screens. I started working on View reservation, then I’ll work on Make reservation, and finally work on Select hotel for User and Manager.

# In addition, each member has contributed to this report:

**Alex**: For this report, I wrote 4. Refactoring as well as added Junit tests for the account classes User and Manager.

**Ethan**: For this report, I wrote the introduction section and wrote the class testing for the reservation class. I also contributed to the validation testing section.

**Minas**: For this report, I was responsible for writing the class testing for the Hotel class.

**Nura**: For this report, I am responsible for writing the class testing for Room Class.

In all, the purpose of this document is to display how we are going about testing and refactoring this project’s code. More details will be addressed in the sections below, which include unit testing, validation testing, and refactoring.

## *Definitions, Acronyms or Abbreviations*

There are a few definitions, acronyms, or abbreviations that are included in this document.

# List of definitions, acronyms and abbreviations:

JDBC - Java Database Connector SQL - Structured Query Language

## *References – Any references used in this document*

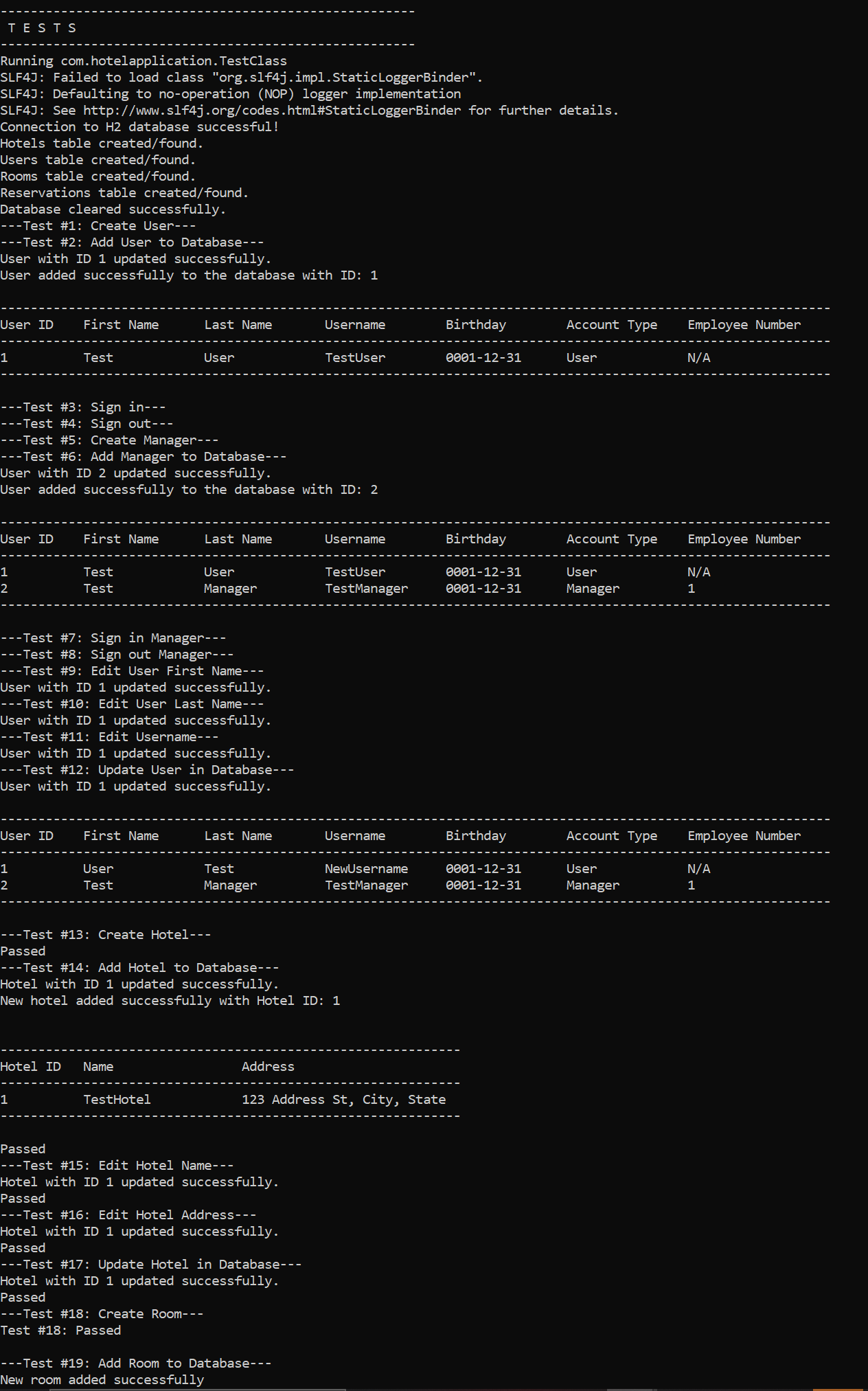
We have not used any references for this paper.

# Note: If you are a group of four then split your group into two pairs. Each pair separately performs number 2 and 3 below. Compile the results in this report and submit.

1. **Unit Testing – (Provide screen shots of testing results)**

You should perform automated unit testing such as use Junit in JAVA, PyUnit/PyTest in Python, etc.

## *Class Testing*



Test #18: Passed

---Test #19: Add Room to Database--­ Nel ' room added successfully

Test #19: Passed

---Test #20: Add Room to Hotel--­ Test #20: Passed

---Test #21: Edit Room Bed Type--­ Test #21: Passed

---Test #22: Edit Number of Beds ir, Room--­ Test #22: Passed

* - -Test #23: Edit Room Price Per Night- - - Test #23: Passed

---Test #24: Edit Room Description--­ Test #24: Passed

---Test #2S: Update Room in Database--­ Room with ID 1001 updated successfully. Test #25: Passed

* - -Test #26: Create Reservation- - - Passed
* - -Test #27: Add Reservation to Database- - -

User: User Test Username: NebUsername Birthday: 12/02/2024

Number of Reservations: 0

Reservation added successfully to the database. Passed

---Test #28: Edit Reservation Start Date--­ Passed

---Test #29: Edit Reservation End Date--­ Passed

---Test #30: Edit Reservation Total Price--­ Passed

* - -Test #31: Update Reservation in Database- - - Reservation l ·ith ID 1 updated successfully.

Passed

Hotel ID Name Address

Ne·,., Hotel Name Ne·, · Address

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Room ID | Hotel ID Room Number | Bed Type | Num of Beds | Price per Night Room Description |
| 1001 |  | twin |  | 130.00 Updated Room Description |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| User ID | First Name | Last Name | Username | Birthday | Account Type | Employee Number |
|  | User | Test | NewUsername | 0001-12-31 | User | N/A |
|  | Test | Manager | TestManager | 0001-12-31 | Manager | 1 |

Reservation ID User ID Room ID Hotel ID Check- In Date Check-Out Date Total Cost

1001 1111-01-10 2222-02-22 150, 00

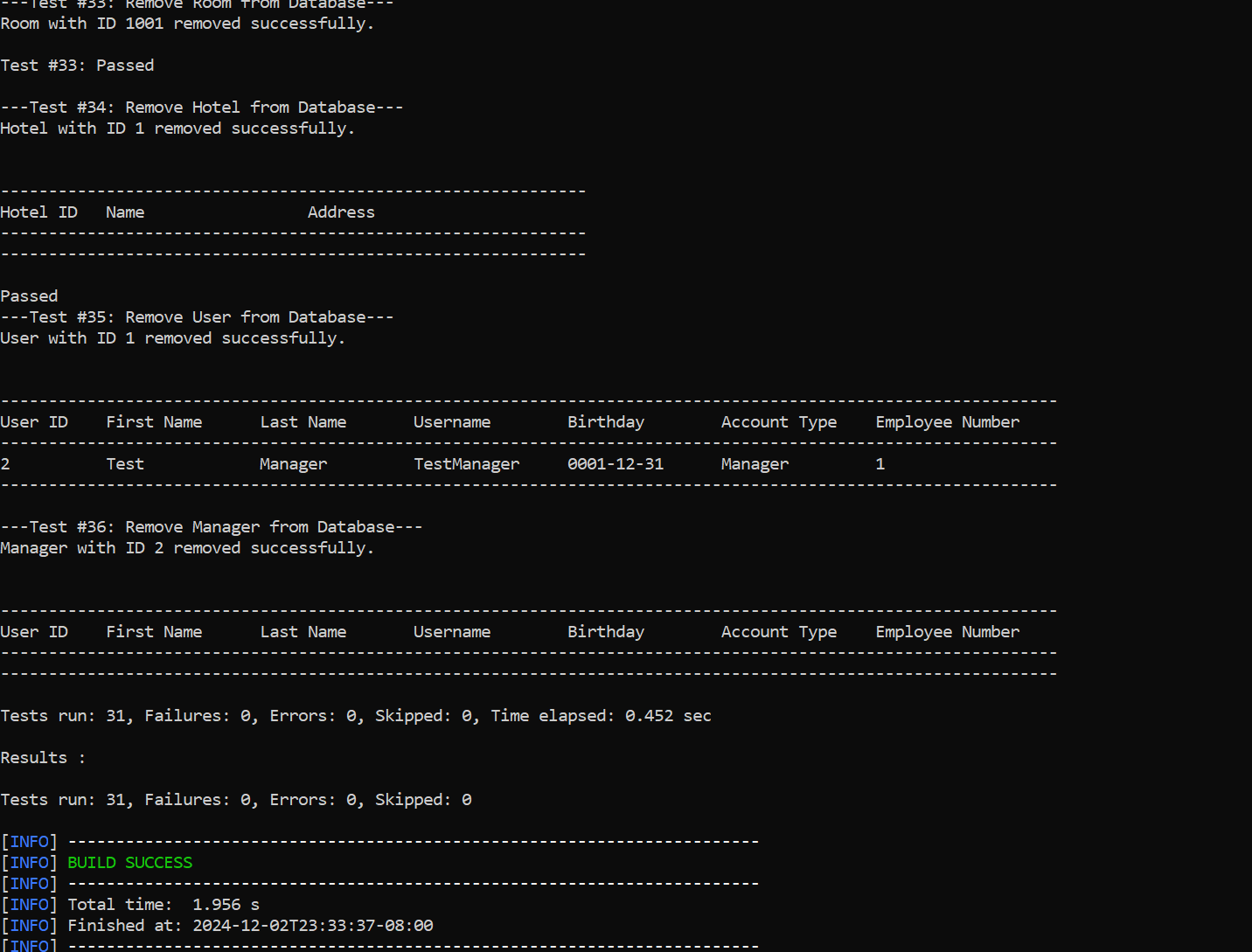
* - -Test #32: Remove Reservation -from Database- - - Reservation , ·ith ID 1 removed success-fully.

Passed

* - -Test #33: Remove Room from Database- - - Room with ID 1001 removed successfully.

Test #33: Passed

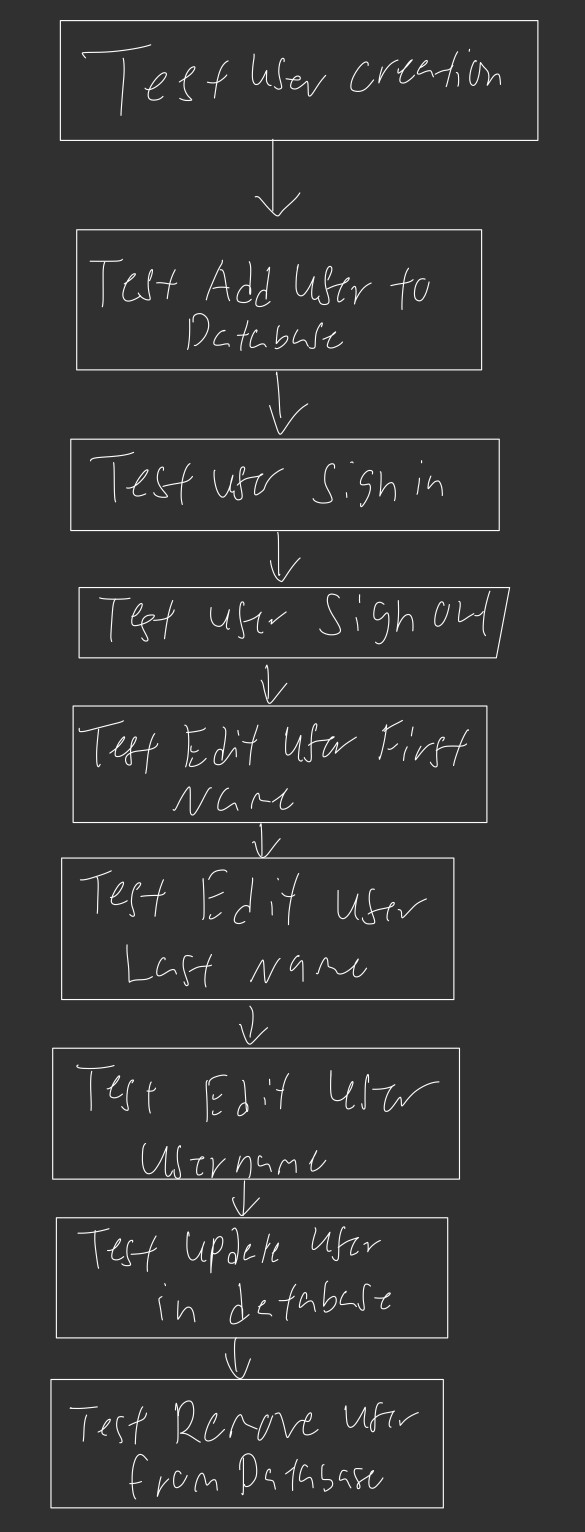
---Test #34: Remove Hotel from Database---



## *Independent Path Testing*

Each pair in the team draws the Flow Graph, determine the independent paths, prepare test cases and compare the results.

## *Flow Graph and Independent paths*



* + 1. ***Test cases and Test results*** – compare the test results with the expected values. If it is difficult to automate such tests then perform manual testing.

a) Test case b) Expected Value c) Test Results d) Conclusion: Passed/Not Passed Test case 1:

1. Test user creation
2. Expected value: user is created successfully
3. Test result: test passed, user is created
4. Conclusion: passed Test case 2:
5. Test hotel creation
6. Expected value: hotel is created successfully
7. Test result: test passed, hotel is created
8. Conclusion: passed Test case 3:
9. Test room creation
10. Expected value: room is created successfully
11. Test result: test passed, room is created
12. Conclusion: passed Test case 4:
13. Test reservation creation
14. Expected value: reservation is created successfully
15. Test result: test passed, reservation is created
16. Conclusion: passed

# Validation Testing

Each pair in the team chooses at least one requirement from Requirement Analysis and performs validation tests. Provide appropriate screenshots of GUI to show the test being carried out.

## *First validation test:*

(3.1) Requirement name: The requirement is allowing users to log in with valid credentials. (3.2) ECs:

1. Valid EC: Username exists, and password matches
2. Invalid EC: Username exists, but password is incorrect / Username does not exist / Empty username or password fields / Password length below the minimum required

(3.3) Test Cases and Test Results:

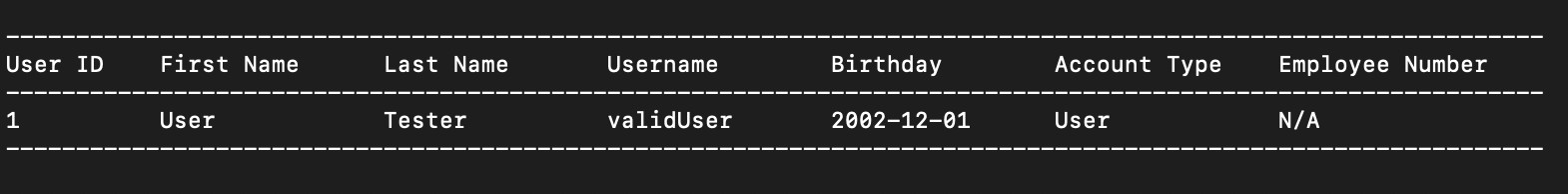
Test case 1:

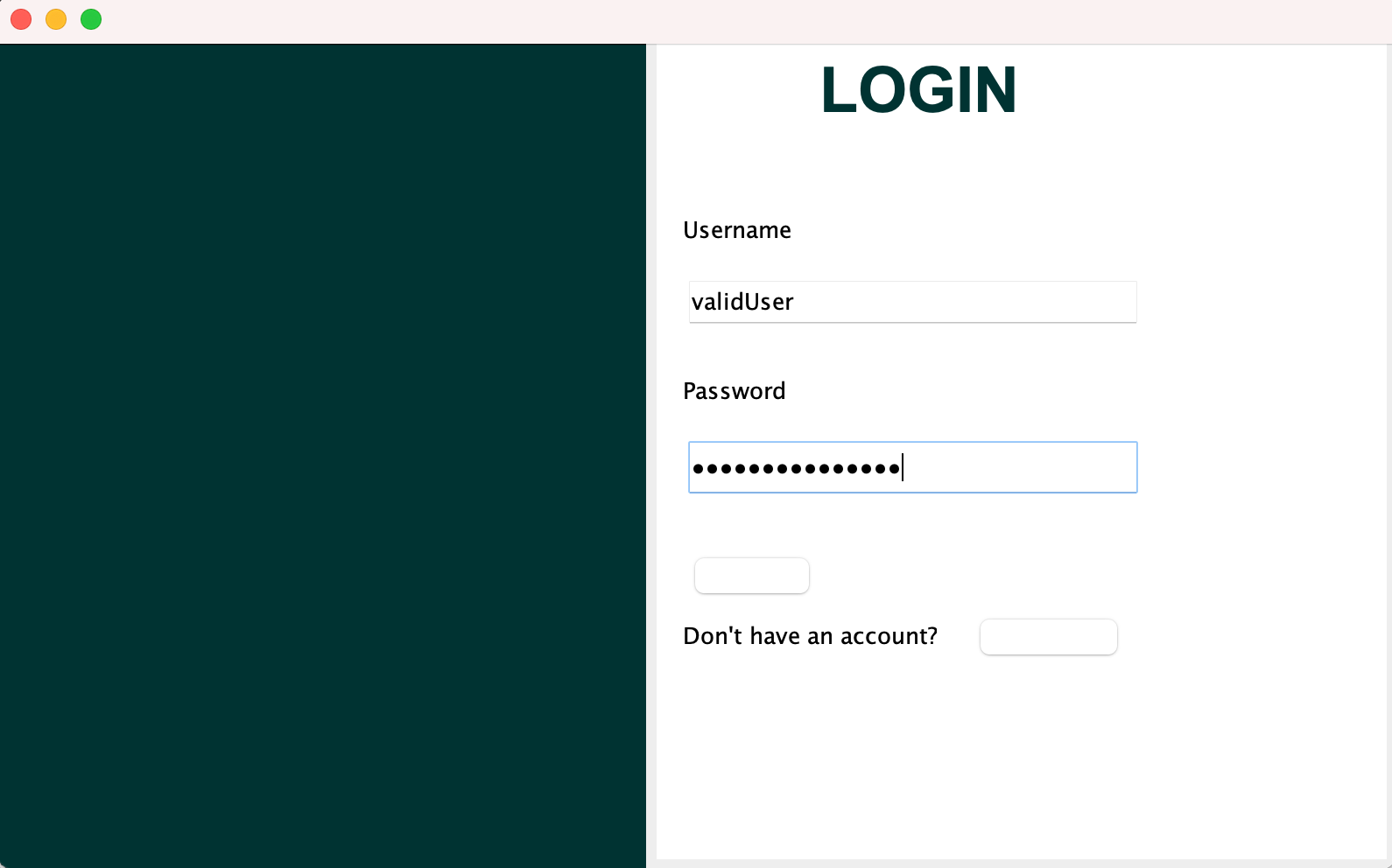
*Input*: Username: validUser | Password: correctPassword

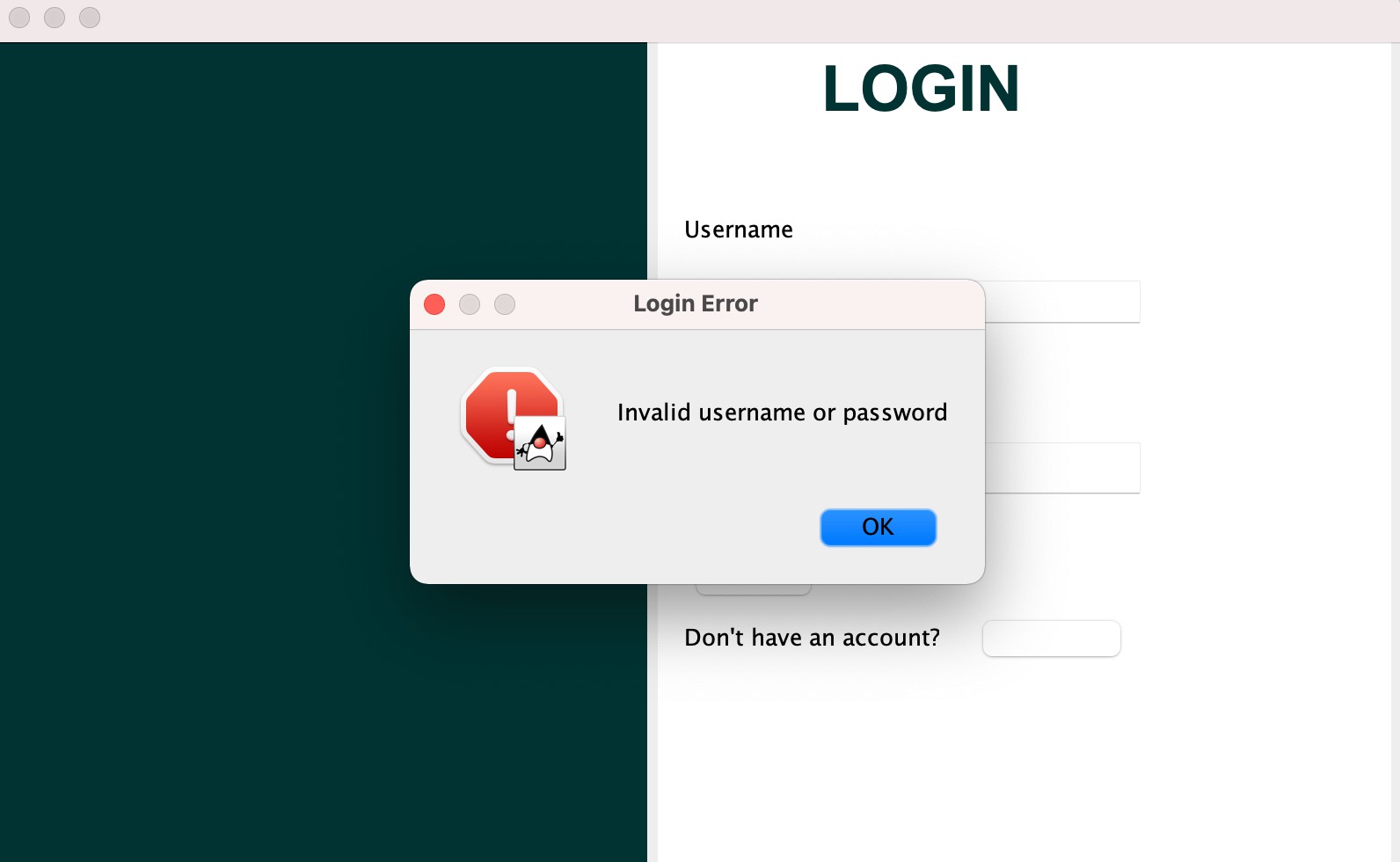
*Expected Output*: User has successfully logged in.

*Actual Output*: Invalid username or password.

*Pass/Fail*: Fail



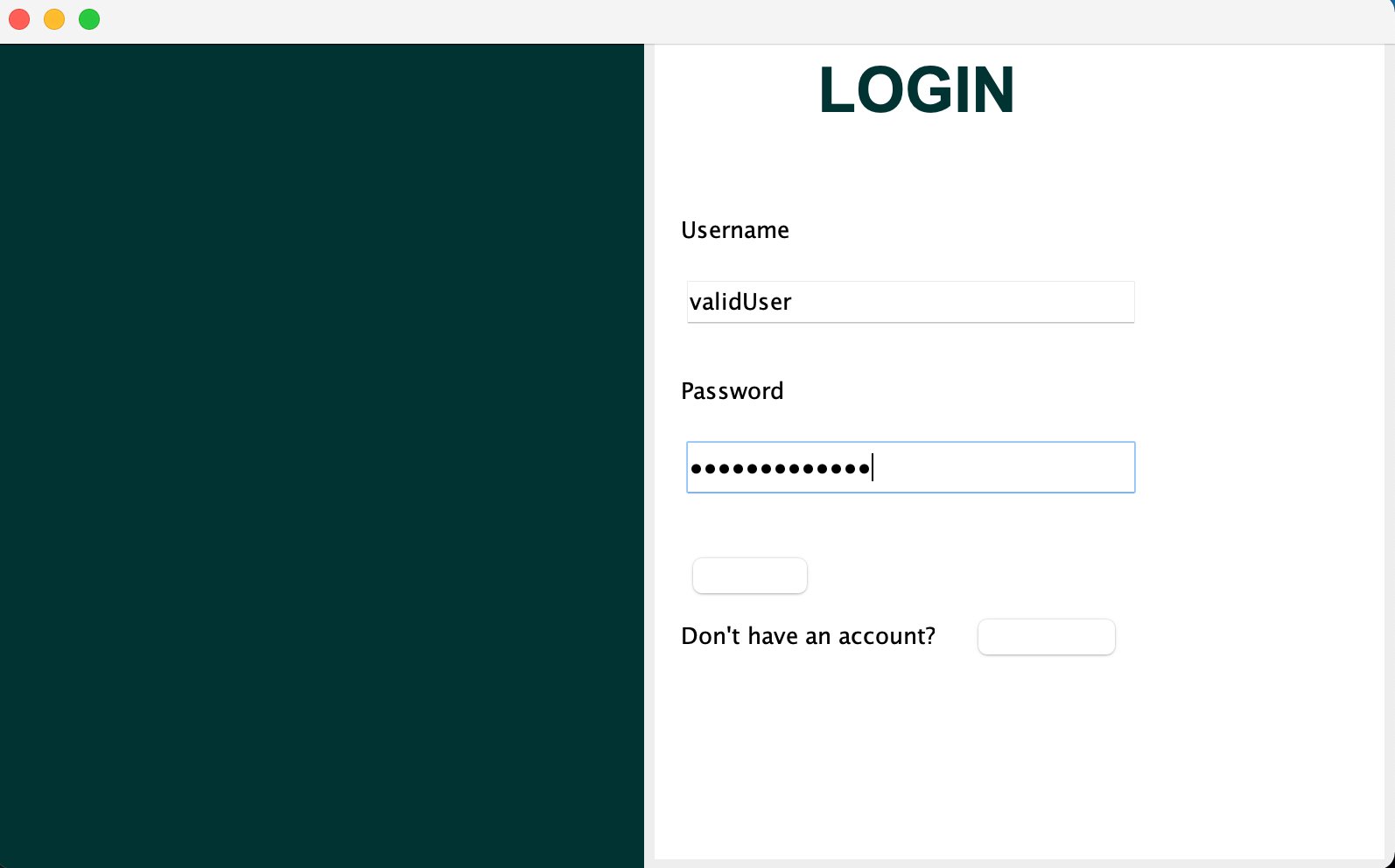


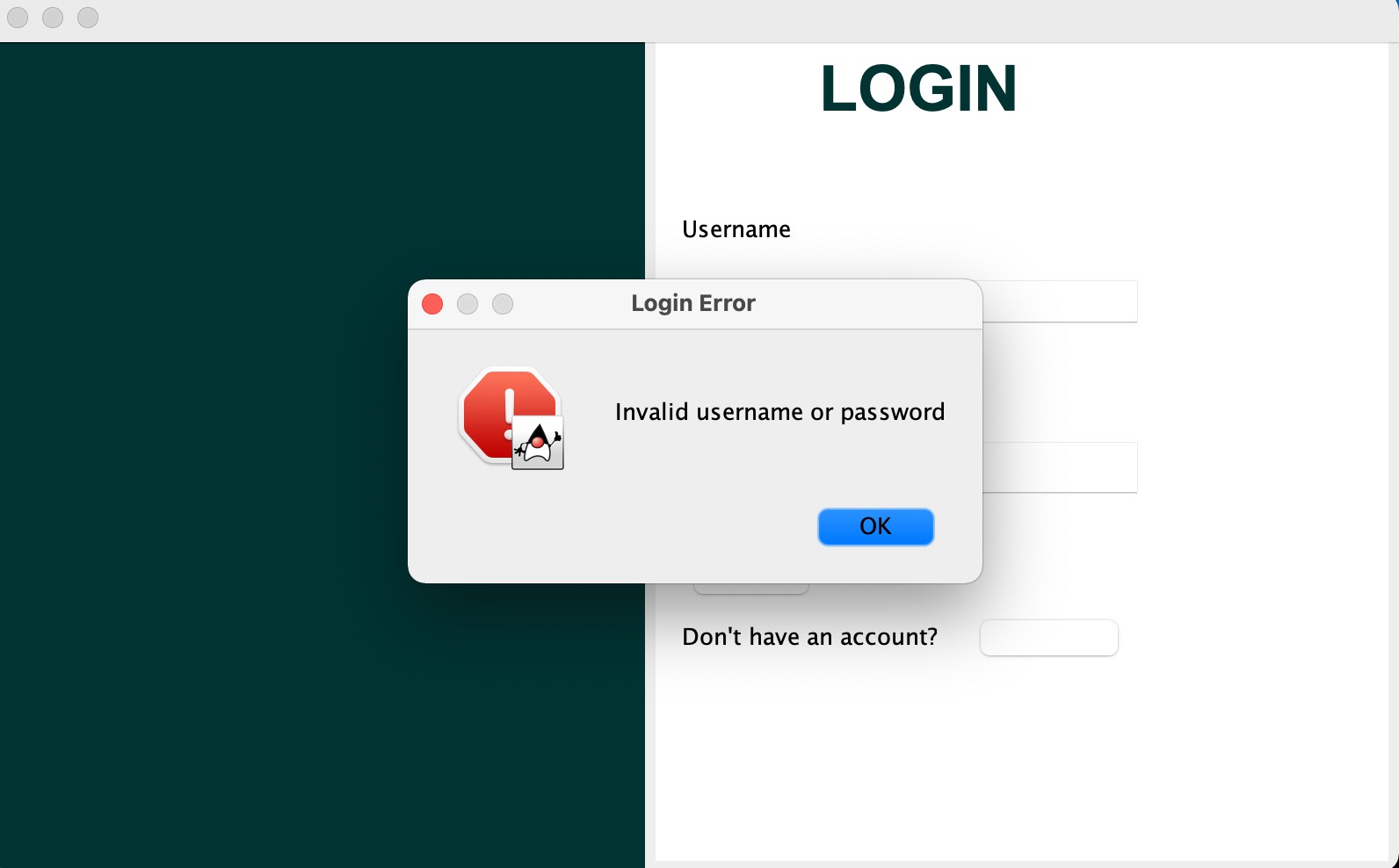


Test case 2:

*Input*: Username: validUsername, Password: wrongPassword *Expected Output*: Error shown saying password is incorrect. *Actual Output*: Error is shown.

*Pass/Fail*: Pass





## *Second validation test:*

(3.1) Requirement name: The requirement is allowing users to create a reservation with a valid start date and a valid end date.

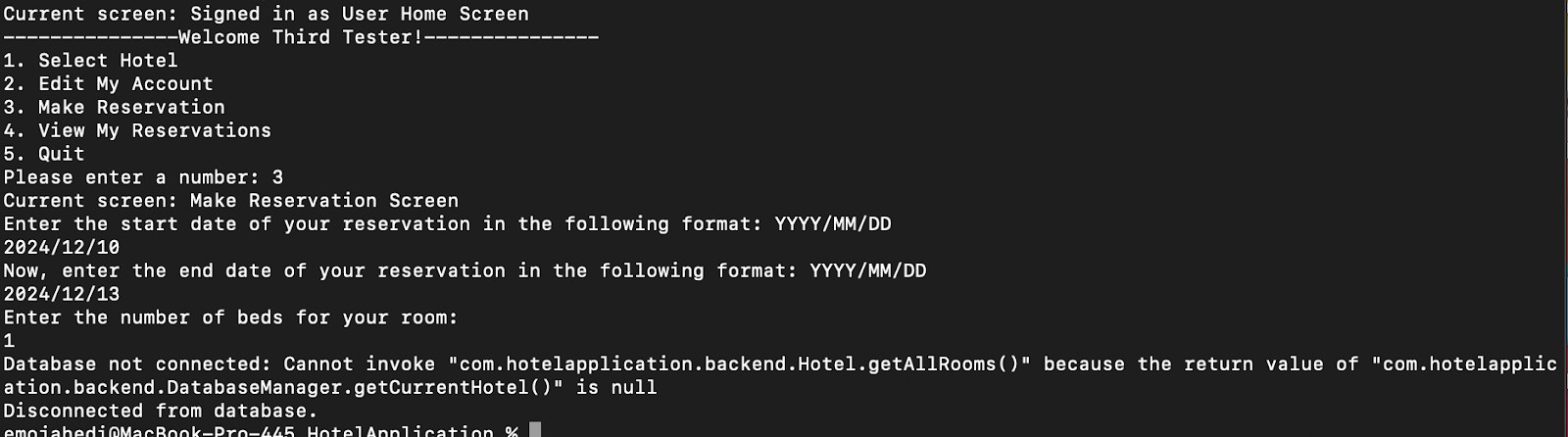
(3.2) ECs:

1. Valid EC: Dates are within range of room availability
2. Invalid EC: Start date is after the end date / Dates are outside room availability range (3.3) Test Cases and Test Results:

Test case 1:

*Input*: Start date: 2024/12/10 | End date: 2024/12/13 *Expected Output*: Reservation was successfully created. *Actual Output*: Error regarding hotel.

*Pass/Fail*: Fail



\*Note: this does work without the database and the standalone application but right now it gives an error.

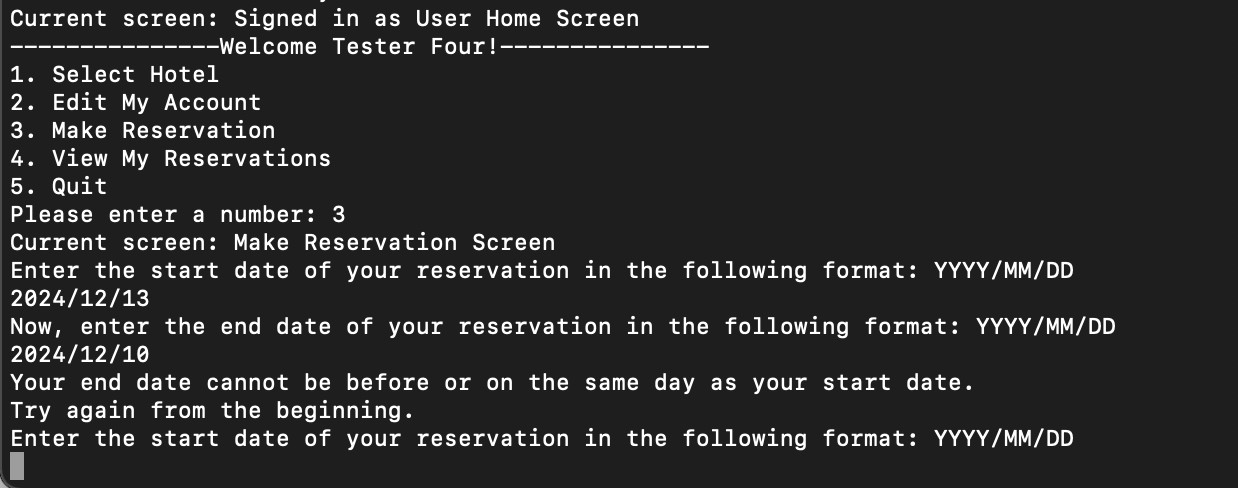
Test case 2:

*Input*: Start date: 2024/12/13 | End date: 2024/12/10

*Expected Output*: Reservation not created, error is shown to prompt user to re-enter.

*Actual Output*: Error is shown on the screen.

*Pass/Fail*: Pass



# Refactoring

Do you recall performing any refactoring to your code/design? If so then give a brief description of the instances along with the reasons.

Yes, we had to refactor to accommodate using the database to store data instead of using lists of objects. The base classes originally created objects and added them to an ArrayList stored in the Database Class. Now they will connect directly to the database via the Database connector.