Final Iteration Report
Hotel Management System

Team name: jigsaw
Shenghao Jiang (sj2914)
Wen Sun (ws2544)
Lingsong Gao (lg3018)
Cheng Shen (cs3750)

In the second iteration demo, Sara finds out that we can add a requirement for a checked-out guest. We need to resolve the confusion on this. Either we should add a requirement for all guests based on their name or we can only add a requirement for the existing guest based on their name and room number.

In the final iteration, the main task is to add input validation to the program and we added a few tests.

Github link
https://github.com/ChengShen1996/ASE

Report Link
https://github.com/ChengShen1996/ASE/tree/master/report

1. CI test

We continue to support continuous integration for your codebase, with both pre-commit and post-commit.

2. Feedback

We took consideration from iteration 1 demo and we added coverage test.

3.User stories and use cases

3.1 revised user story

User story 1: As a manager, I want to see the revenue so that I can evaluate the operation status of the hotel and set room price. My condition of satisfaction is that I can record daily profits and outputs and visualize our data for analysis and comparison.

User story 2: As front desk, I want to be able to help customers check in and check out so that we have the right information. My condition of satisfaction is that I can check in and check out our customers on our software interface and the availability status of the room will be changed accordingly.

User story 3: As front desk, I want to be able to tell the customers our available rooms and the prices, so that I arrange the most suitable rooms for them. My condition of satisfaction is that the information about our available rooms is correct and can be shown on our software interface.

User story 4: As front desk, I want to be able to record requirements of customers and add other room service for our customers, so that customers can enjoy room service and hotels can make profits. My condition of satisfaction is that the information can be recorded on the interface.

User story 5: As front desk, I want to be able to recommend restaurants, museums, and other places of interests around our hotel, so that customers can go to desirable places. My condition of satisfaction is that the information should include lists of places according to some criterion.

3.2 use cases

Use case 1:
Title: Check the daily revenue of the hotel

Actor: Manager of the hotel (Registered User)

Description: Part of the hotel management system, the manager clicked the manager button and then clicked the hotel revenue button, then he can see the operations in the hotel, like what room has been booked and the profit the hotel got from it.

Preconditions: the hotel is in operation, there has been guests check in and check out before.

Triggers:The manager clicked the Manager and then clicked hotel revenue.

Basic flow:
1. The system provides a front page with a button front desk and manager on it. If the user if the manager and wants to see the revenue, the user should click the button manager. Then the system will turn to another page with two buttons saying hotel revenue and room price on it. If checking the hotel revenue is what the user wants, then the user should click button check revenue.
2. When check revenue is clicked, the system will pop out another window requiring the user to input year, month. After the user input the year and month of the revenue he wants to see, the user should click ok.

3.After the user clicked ok, the system will go query the database. The system will make a sum over the specific year and month and return the revenue the hotel made during this month. The system will show this result to the user on the interface.

Alternate flow: for this use case, this is the only way to see the revenue, thus there is no other alternate flow for this problem.

Use case 2:
Title: Front desk daily work

Actor: Front desk (Registered User)

Description: This is the main function for a hotel to sustain its daily business. The hotel needs a system to keep track of which room is being taken and how long it's been taken and the total price. As for the front desk, the system should first be able to provide all the available rooms to the customer, then commit check in for the customer. And when the customer is leaving, front desk should be able to check customers out of the hotel. The front desk also need to update the room status when the customer checks out and provide the correct total check out price for the customer.

Preconditions: The information about room type and room price is set up.

Triggers: When the guest picked a room and wants to check in, the checking in function will be triggered. When the guest require to check out, the check out will be triggered.

Basic flow:
1. First the user should choose Front Desk page at the log in window, then the system will turn to a page for front desk users.
2. At the front desk page available room section, the user should first input the start date and end date, then click ok, the system will go query the database and get the room type and room number for all the rooms that are available during this period.
3.When the customer has picked a certain room, then the front desk staff can commit check in. The user should click guest+ button. After this button has been clicked, the system will pop out a window with three buttons on it. To check in, the user should click the check-in button.
4. After clicking the check-in, the system will pop out the window for the user to write the information need for check-in including guest name, room number, check-in data, check out data, and the requirements. Also, there this a button on the page that says click to show total price. This will show the user how much it would cost to stay in a certain room for the period of staying time. If all the information has been written, the user can click save to commit check-in function. Then the system will then insert all the information to the database.
5. If the guest wants to check out, the user can click Check-Out button.

6. After clicking the checkout button the system will pop out a window with some input space on it including guest name and room number. The user should input both to complete check out. If the input guest name and room number are correct, then the customer will be check out successfully.

Alternate flow: for this use case, this is the only way to do check in and check out, thus there is no other alternate flow for this problem.

Use case 3:
Title: Front desk daily work-check for available rooms

Actor: Front desk (Registered User)

Description: This is the main function for a hotel to sustain its daily business. The hotel needs a system to keep track of which room is being taken and how long it's been taken and the total price. As for the front desk, the system should be able to provide all the available rooms to the customer.

Preconditions: The information about room type and room price is set up.

Triggers: When a new guest come to the front desk and ask for available rooms.

Basic flow:
1. First the user should choose Front Desk page at the log in window, then the system will turn to a page for front desk users.
2. At the front desk page available room section, the user should first input the start date and end date, then click ok, the system will go query the database and get the room type and room number for all the rooms that are available during this period.
3.Then the system will show all the available rooms in the format of a table, including room type and room number.

Alternate flow: for this use case, this is the only way to do check in and check out, thus there is no other alternate flow for this problem.

Use case 4:
Title: Extra service for guests
Actor: Front desk (Registered User)

Description: During hotel operation, there will come a time when hotel guests would have other requirement like morning wake up or breakfast. So this part of the program will help hotel management to record all these extra requirements.

Triggers: When an existing guests made a requirement and then the front desk staff will have to trigger this operation.

Basic flow:

1. First the user should choose Front Desk page at the log in window, then the system will turn to a page for front desk users.
2. At the front desk page guest information section, the user should first input the name of the guest and the room number, then click ok, the system will go query the database and get the information about the guest including name check-in data check-out date and total price. There will also appear a requirement block where the front desk staff can input the requirement of the guest.
3. After the user input the requirement of the guest and clicked ok, the system will go to the database and update the information about this guest.

Alternate flow: for this use case, this is the only way to do check in and check out, thus there is no other alternate flow for this problem.

Use case 5:

Title: Recommendation for guests

Actor: Front desk (Registered User)

Description: Often there will come a time when hotel guests would need advice from the local to know about the nearby restaurants and places of interest like museums and galleries. This part of the program will search these result for the guest.

Triggers: When an existing guests ask for recommendation and the hotel staff use the software's recommendation function, this part will be triggered.

Basic flow:

1. First the user should choose Front Desk page at the log in window, then the system will turn to a page for front desk users.
2. At the front desk page recommendation section, the user should first the type of recommendation they want, we have provided the following choices: restaurants, arts, food, nightlife, shopping. The user can also specify their request by choosing a subtype.
3. After the type and subtype has been chosen, the user can click the button Search at the bottom.
4. Then the system will go use the yelp API to get the result and then put the results in the fomat of a table to present to the user.

Alternate flow: for this use case, this is the only way to do check in and check out, thus there is no other alternate flow for this problem.

4.Test plan

4.1 API

In the API test, we want to test the response code and returned results from API call. We made our tests based on different parameters, such as type, subtype, and location.

Valid equivalence partition:
type = “restaurants”, subtype = “buffets”, location = “NYC”, which returns response code 200, which means success. The result is not empty.
type = “restaurants”, subtype = “food”, location = “NYC”, which returns response code 200, which means success. The result is not empty.

Invalid equivalence partition:
type = “res”, subtype = “aaaaa”, location = “NYC”, which returns response code 200, which means success. However, the result is empty.

These tests are in apiTest.

4.2 Database

We first test all basic database operations: select, insert, update, together with databaseHandler constructor.
Test insert into database
Valid equivalence partition: Insert

```
String name = "test-name2";
String roomId = "101";
String checkInDate = "2012-12-12";
String checkOutDate = "2012-12-13";
String requirement = "burger";
String totalPrice = "100";
```

These input will successfully insert into the database and we found these values in the database.

Invalid equivalence partition:
If the roomId is not integer, the insertion will fail.
If checkInDate or checkOutDate are not in the correct date format, the insertion will fail.
The test is in selectDatabaseTest.

Boundary condition: when checkInDate equals to checkOutDate

Test update database:

Valid equivalence partition: we tried to update the isGone attribute to be true
The result shows the isGone value is true
inValid equivalence partition: we tried to update the isGone attribute to be false
The result shows the isGone value is false
The test is in updateDatabaseTest.

Test select from database:
We select the record where name = “test-name2” and roomId = “101”, and we successully found the matched record.

4.3 UI Interface Test

We have UI tests for each individual UI, such as checkInUITest, checkOutUITest, guestMenuUITest, logInUITest, roomServiceUITest and so on.

4.4 Controller Classes (all tests below are in the tests folder)

4.4.1 ChckInTest

We tried to test the login controller

Valid equivalence partition:
Guest name = “chaiquan”
Room Number = 101
Check-in Date = 2018-11-11
Check-out Date = 2018-11-13

Invalid equivalence partition:
Room Number = 999
Room Number = -1
Check-in Date = 1
Check-out Date = 2
Check-in Date = “x”
Check-out Date = “b”
Check-in Date = 2011-11-11 and Check-out Date = 2011-11-09 (check out date is earlier than check in date)

Boundary condition: when checkInDate equals to checkOutDate

4.4.2 CheckOutTest

Valid equivalence partition:
Guest Name = “a”
Room Number = 101

Invalid equivalence partition:
Room Number = 999
Room Number = -1

4.4.3 FrontDesk available rooms test

Valid equivalence partition:
start date = 2012-12-12
end date = 2012-12-14

Invalid equivalence partition:
start date = 1
end date = 2
start date = 2012-12-13 and end date = 2012-12-12 (end date is earlier than start date)

Boundary condition: when start date equals to end date

4.4.4 Guest information test

Valid equivalence partition:
Name = “a”
Room Number = 101

Invalid equivalence partition:
Room Number = 999
Room Number = -1

4.4.5 RecommendationTest

Valid equivalence partition:
Type = “arts”
Subtype = “galleries”
Type = “restaurants”
Subtype = “buffets”

Invalid equivalence partition:
Type = “aaaaa”
Subtype = “bbb”
Note: In order to prevent user from entering invalid input, we designed dropdown menu.

4.4.6 ShowRevenueTest

Valid equivalence partition:
Year = 2014
Month = 1
Year = 2018
Month = 10

Invalid equivalence partition:
Year = 2020
Year = -1
Month = 13
Month = 0
Note: In order to prevent users from entering invalid input, we designed dropdown menu.

4.4.7 setRoomPriceTest

Valid equivalence partition:
Single = 150
Double = 200
Suite = 300

Invalid equivalence partition:
Single/Double/Suite = -1

Boundary condition: Single/Double/Suite = 0

4.4.8 RoomServiceTest

Valid equivalence partition:
Guest Name = “a”
Room Number = 101
Requirement = “I want a burger”

Invalid equivalence partition:
Room Number = 999

Room Number = -1

5.Coverage

We measured our branch coverage with Intellij locally and use coverall in the post-commit CI test.
For post-commit CI, we set up coveralls on travis ci, right now we have achieved 75.28%. The reason we did not reach a very high coverage is that each scene contains at least start() and main(), which can not be easily tested by our tool. We will try to reach higher level of coverage in the final iteration.
The detailed coverage report is in the report folder.