Project Design:

Hotel Reservation Program

Group 8:

Jules Torres, Megan Moore, Mario Bethancourt

CMSC 495

Section 7380

CMSC 495/7380 Capstone in Computer Science

November 14, 2023

Professor Hung Dao

Revision History

Date	Description	Author
11/9/2023	Created first draft of the project plan	Mario Bethancourt
11/12/2023	Updated project design Wrote pseudocode and scenarios	Megan Moore
11/14/2023	Revised pseudocode and final revisions	Jules Torres, and Mario Bethancourt
11/19/2023	Made changes based on the feedback	Mario Bethancourt
12/09/2023	Made final updates	Mario Bethancourt

Project Design Overview

The Hotel Reservation System is a Java program that allows users to book hotel rooms. It provides an easy and efficient way for individual clients to plan and manage their hotel reservations from their devices.

Objectives:

1. Create a user-friendly interface for browsing available hotels.
2. Implement functionalities that allow the user to input their desired room from a list of rooms, they can input their check in and check out date, and how many people will be staying in the room.
3. Enable users to view detailed information about each room within the hotel, including room types, number of beds, and prices.
4. Provide a login system that saves the username and password allowing users to log into the system.
5. Implement a booking system that uses the current date, allowing users to only reserve rooms on future days..
6. Allow users to view their reservations, and allow them to cancel reservations.
7. Integrate a payment processing system that gives the user the total price per night for their reservation.

Class Diagram

The system contains the following classes:

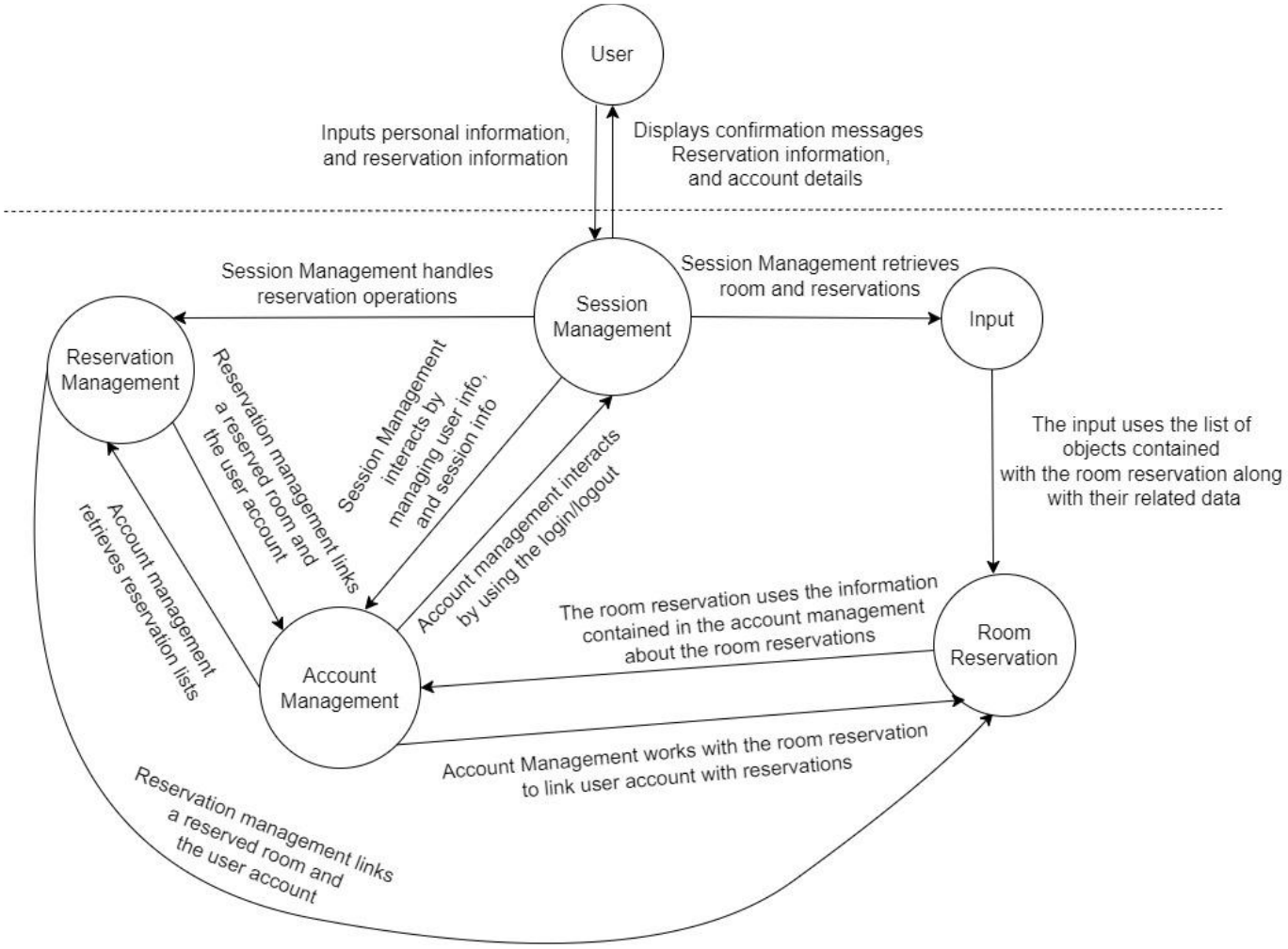
Input: This subsystem is responsible for initializing and managing the underlying data structures for the hotel reservation system. It creates and maintains a collection of RoomReservation objects representing the available rooms in the hotel. Additionally, it manages the input data relevant to the hotel, such as the hotel name and the number of floors. The Input class provides a foundation for the hotel reservation system by organizing and handling essential data structures and information.

Account management: This subsystem represents user accounts within the hotel reservation system. It encapsulates user details such as name, contact information, and login credentials. Additionally, it manages reservations associated with each user, providing methods for updating user information and handling password verification. This class is integral to user authentication, account creation, and account updates, ensuring secure and accurate management of user data.

Room Reservation: This subsystem represents individual hotel rooms, storing information about their occupancy status and associated reservations. It contains methods for occupying and vacating rooms, allowing for dynamic updates to room availability. This class is essential for tracking the status of each room, ensuring accurate information is displayed to users during the reservation process and in the main menu.

Session management: This subsystem serves as the central control unit for managing user sessions in a hotel reservation system. It encapsulates the functionality for user authentication, handling reservations, updating user accounts, and managing check-in and check-out processes. This class orchestrates the user interface, using Swing components to display various panels and dialogs. It also interfaces with the underlying data structures and files to load and save user information, providing a comprehensive structure for managing user interactions within the hotel system.

Reservation management: This subsystem models individual hotel room reservations, capturing details such as check-in and check-out dates, the number of occupants, and associated costs. It is closely tied to the AccountManagement class, linking reservations to specific users. This class facilitates the booking process, updates the availability of rooms, and calculates reservation costs. It plays a crucial role in maintaining a record of user reservations and their corresponding details.

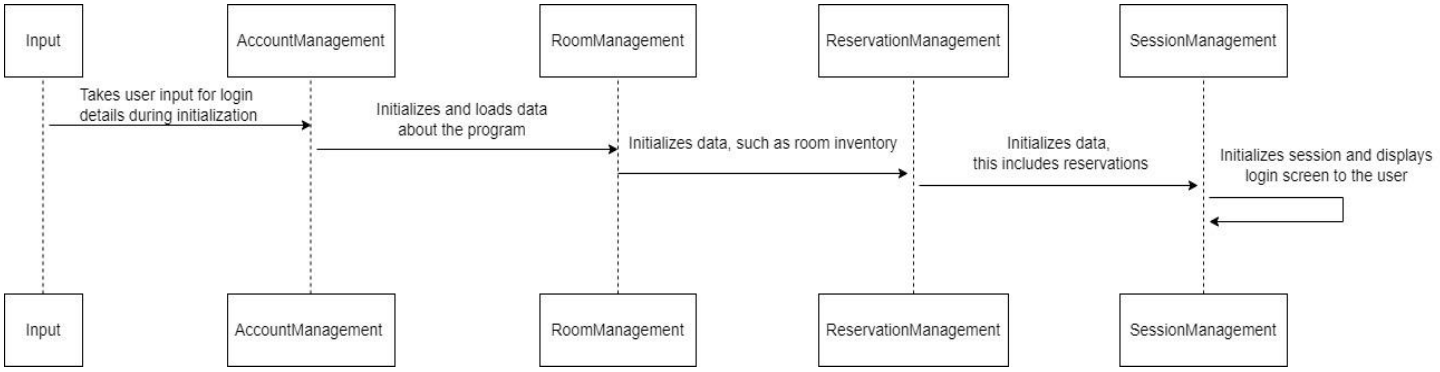


Event-trace diagram:

Scenario 1: Startup
Description: Initializes the hotel reservation system, loads all necessary data. Upon starting the program, the user will see the main menu displayed on their device asking for login details or to create an account.

Precondition: The user opens the application, then the program is running and ready for the user to login.

Post-Condition: The subsystems are initialized, and the login screen is displayed. The user will login with their credentials and continue to book a hotel room.

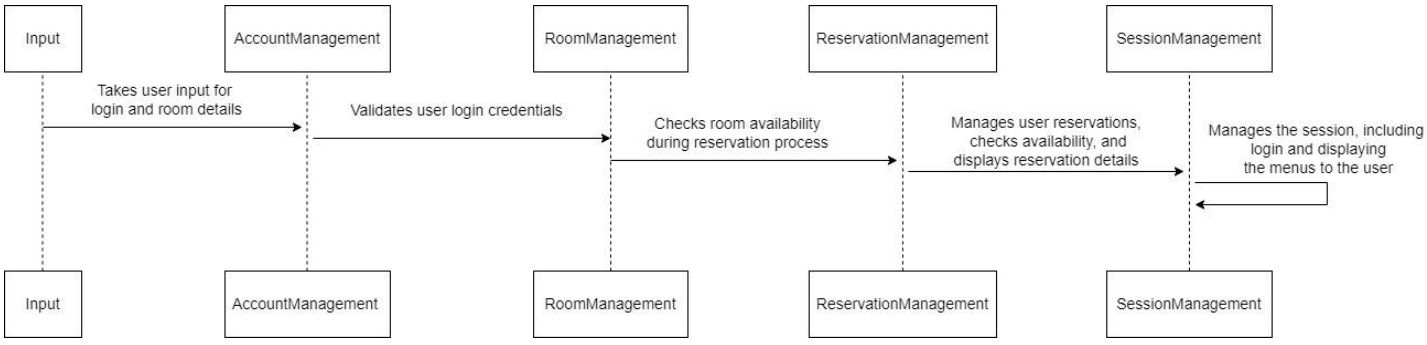


Scenario 2: Normal operation

Description: During this scenario the main functionalities of the hotel reservation system can be accessed by the user. Upon user logging into the system, the user will see the Hotel Reservations Dashboard and continue to search for rooms to book through the system on their device.

Precondition: The program is running, and the user is logged in and has access to the main menu and all the system's functionalities.

Post-Condition: Depending on the chosen operation, the system performs the requested action. The user will search and book a hotel room, a confirmation email is sent and displayed on booking.

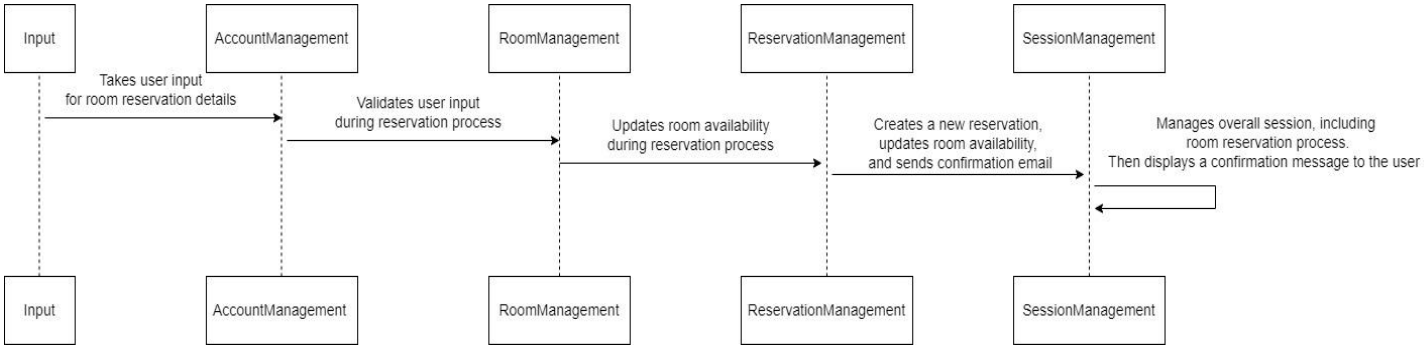


Scenario 3: Reserving a room

Description: The user decides to make a reservation. The user navigates to the reservation section, selects the desired dates, room type, and any additional preferences. The system checks for room availability and confirms the reservation.

Precondition: The program is running, and the user is logged in.

Post-Condition: A reservation is made, and the system updates the room availability. A confirmation email is sent to the user with reservation details.

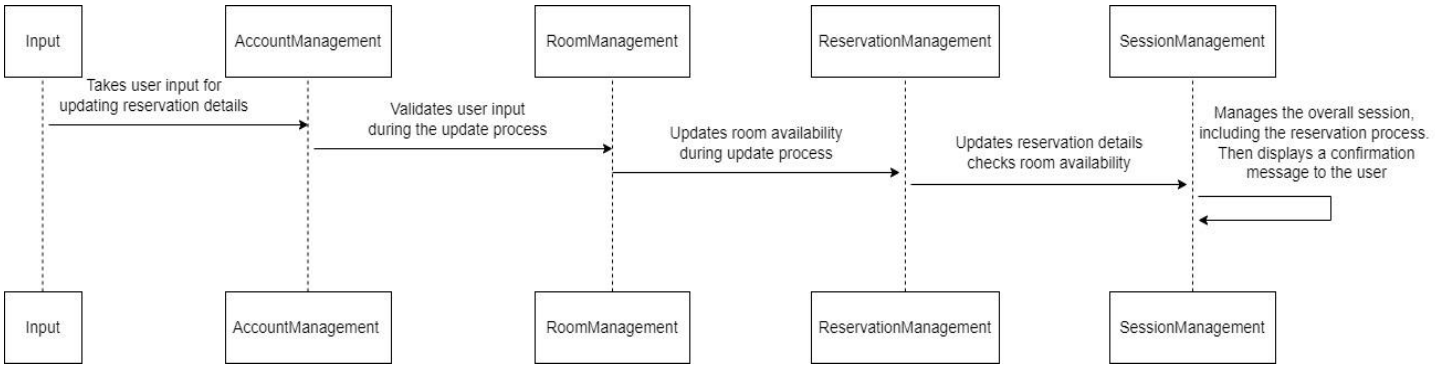


Scenario 4: Updating a reservation

Description: The user wants to make changes to an existing reservation, such as modifying dates or room preferences. The user navigates to the reservation management section, selects the reservation to update, and modifies the necessary details.

Precondition: The program is running, the user is logged in, and the user has an existing reservation.

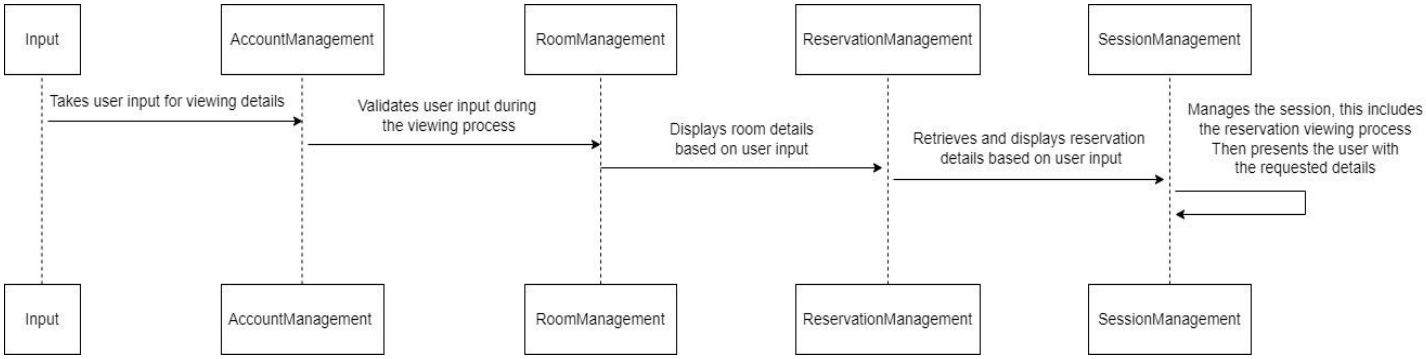
Post-Condition: The reservation is updated, and the system sends a confirmation email with the updated reservation details.



Scenario 5: Viewing reservations
Description: The user wants to view the details of a specific reservation. The user navigates to the reservation history section, selects the desired reservation, and views the details such as dates, room type, and total cost.

Precondition: The program is running, the user is logged in, and the user has existing reservations.

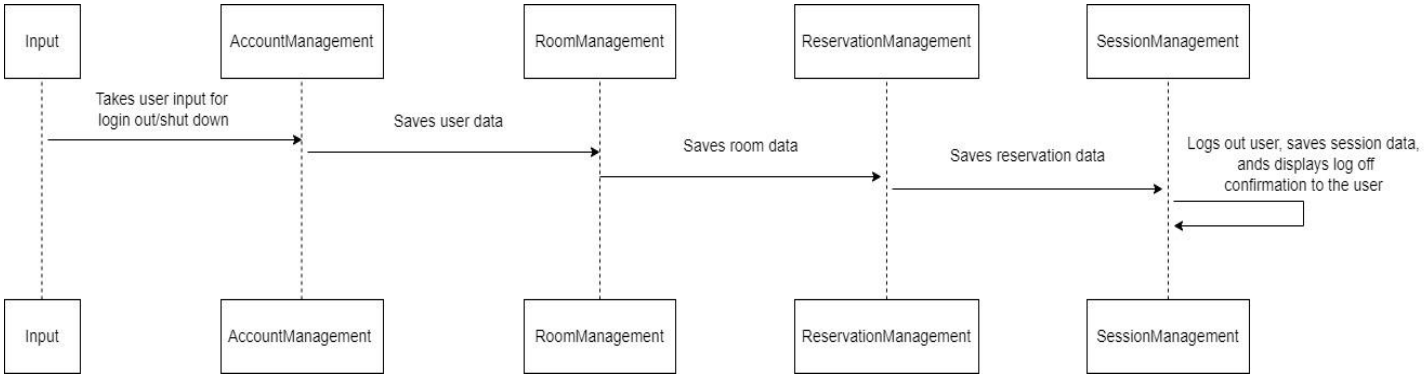
Post-Condition: The user views the reservation details, and the system displays the relevant information without modifying the reservation.



Scenario 6: Shut down
Description: The user initiates the log-off process. Once a user books a hotel room, the user will log out of the system. The user data is saved, and a log-off confirmation message is displayed.

Precondition: The program is running, the user is logged in and the hotel is booked.

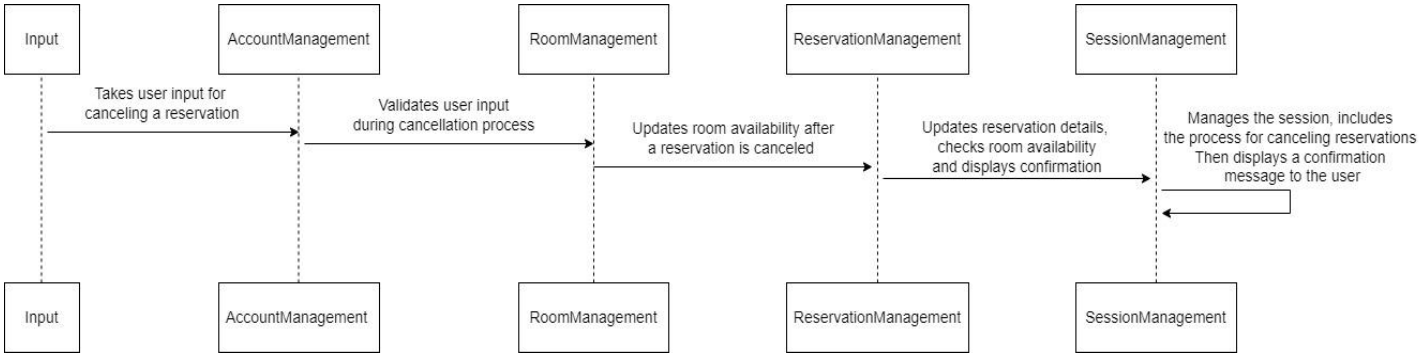
Post-Condition: The user logs out of the system after booking or the system will automatically log the user out after X amount of time. User data is saved, a log-off confirmation message is displayed.



Scenario 7: Cancel Reservation
Description: User decided to cancel or alter reservation with hotel.

Precondition: The program is running, the user is logged in and the hotel is booked.

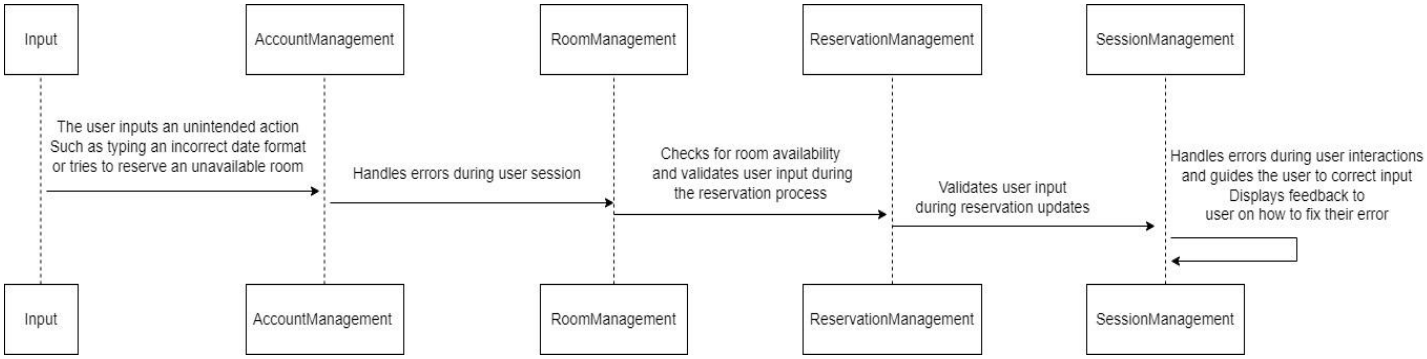
Post-Condition: Reservation is canceled or altered by user, a confirmation email is sent and displayed.



Scenario 8: Error handling
Description: This scenario occurs during a user interaction that did not go as intended. The system checks for errors, displays error messages if necessary, and guides the user to correct input. Examples of unintended actions:The user enters the date in an incorrect format, the user attempts to reserve a room that is unavailable, or the user leaves required information black such as the number of guests or valid user information.

Precondition: The user initiates an action that requires input, such as making a reservation or updating information.

Post-condition: If an error is found, an error message is displayed, guiding the user to correct their input. If no error is found, the system performs the requested action, and the user continues with normal operations.



Class Design:

In this section, pseudocode is listed for the Hotel Reservations System.
Hotel Reservation System Pseudocode:

SessionManagement.java

```
import javax.swing.*;
import javax.swing.border.Border;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.HashMap;
import java.util.Map;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import java.io.*;
import java.util.Random;

class SessionManagement:
    properties:
        - currentUser: AccountManagement
        - users: Map<String, AccountManagement>
        - hotelInput: Input
```

```
methods:
    method handleCreateAccount():
        // Handles the creation of a new user account

    method handleDeletion(roomNumber: String):
        // Handles the deletion of a reservation for the current user

    method handleLogin():
        // Handles the login process for a user

    method handleLogout():
        // Handles the logout process

    method updateAvailableRooms():
        // Updates the available rooms information in the UI

    method handleReservation():
        // Handles the reservation process for the current user

    method handleUpdateAccount(...):
        // Handles the update of user account information

    method handleCheckIn():
        // Handles the check-in process for the current user

    method handleCheckOut(roomNumber: String):
        // Handles the check-out process for the current user

    method generateRandomID():
        // Generates a random check-in ID

    method updateReservationAvailability():
        // Updates the reservation availability information in the UI

    method updateUserAccount():
        // Updates the user account information in the UI

    method loadUsersFromFile():
        // Loads user information from a file
```

method saveUsersToFile():
// Saves user information to a file

method main(args: String[]):
// Entry point of the application

Input.java

import java.util.ArrayList;
import java.util.List;

Class Input:
Properties:
- List<RoomReservation> rooms

Methods:
- Constructor(name: String, floors: int)
Initialize the Input instance with an empty list of rooms. Create rooms for each floor and add them to the list.

- createRoom(floor: int, type: String): void
Create a new room with a unique room number based on the floor and type. Add the room to the list of rooms.

- convertTypeToNumber(type: String): int
Convert the room type to a corresponding number. Used for room number generation.

- getRooms(): List<RoomReservation>
Return the list of rooms in the hotel.

- getRoomByNumber(roomNumber: String): RoomReservation
Return the RoomReservation object corresponding to the given roomNumber.

AccountManagement.java

import java.util.ArrayList;
import java.util.List;
import java.io.Serializable;

Class AccountManagement:

Properties:
- String name
- String lastName
- String phoneNumber
- String streetNumber
- String streetName
- String email
- String state
- String city
- String zipCode
- String username
- String password
- String checkInID
- List<ReservationManagement> reservations

Methods:
- Constructor(name: String, lastName: String, phoneNumber: String, streetNumber: String, streetName: String, email: String, state: String, city: String, zipCode: String, username: String, password: String): AccountManagement
  Initialize an AccountManagement instance with the provided user information and an empty list of reservations.

- Constructor(username: String, password: String): AccountManagement
  Initialize an AccountManagement instance with only a username and password.

- updateUserInfo(name: String, lastName: String, phoneNumber: String, streetNumber: String, streetName: String): void
  Update the user's information with the provided details.

- Setters and Getters for name, lastName, phoneNumber, streetNumber, email, state, city, zipCode, streetName, username, checkInID, password, reservations.

- checkPassword(password: String): boolean
  Check if the provided password matches the user's password.

- addReservation(reservation: ReservationManagement): void
  Add a reservation to the list of reservations.

- removeReservationByRoomNumber(roomNumber: String): void
  Remove a reservation from the list of reservations based on the room number.

- updatePassword(oldPassword: String, newPassword: String): boolean
  Update the user's password if the old password matches; return true if successful, false otherwise.

- equals(obj: Object): boolean
  Override equals method.

- hashCode(): int
  Override hashCode method.

RoomReservation.java

Class RoomReservation:

Properties:
- String roomNumber
- boolean occupied
- AccountManagement occupant
- String type

Methods:
- Constructor(roomNumber: String, type: String, price: double): RoomReservation
  Initialize a RoomReservation instance with the provided room number, type, and price.

- Constructor(roomNumber: String): RoomReservation
  Initialize a RoomReservation instance with the provided room number.

- Setters and Getters for roomNumber, type, occupied, occupant.

- occupy(occupant: AccountManagement): void
  Mark the room as occupied by the given occupant.

- vacate(): void
  Mark the room as unoccupied.

- getPricePerNight(): double

Return the price per night based on the room type.

ReservationManagement.java

import java.time.LocalDate;
Class ReservationManagement:

Properties:
- LocalDate checkInDate
- LocalDate checkOutDate
- double roomPrice
- RoomReservation reservedRoom
- RoomReservation room
- int numberOfPeople

Methods:
- Constructor(room: RoomReservation, occupant: AccountManagement, checkInDate: LocalDate, checkOutDate: LocalDate, numberOfPeople: int): ReservationManagement

Initialize a ReservationManagement instance with the provided room, occupant, check-in date, check-out date, and number of people.

- Setters and Getters for roomNumber, checkInDate, checkOutDate, reservedRoom, room, roomPrice, and numberOfPeople.

Unresolved risks and risk mitigation:

The risks are still the same from the analysis. There isn't enough time to implement the mitigations, so the old risks remain the same as of now, with a new risk included:

- Data security risk: Storing personal information requires robust data security. Currently we are storing user information on a text file which is not the most secure option. To mitigate this risk, we can implement a lockout system that limits the number of password attempts for users in the future. Additionally, implementing encryption and industry-standard practices such as two factor verification, can enhance data protection. All of this can not be implemented as of now due to time restraints.
- Data privacy risk: Customers may have concerns about the confidentiality of their personal information. Adding a privacy policy can reassure users of their safety and privacy.
- Data loss and recovery: Data loss may occur if the system crashes.To mitigate this the user data will be regularly backed up.

- Software integration: the program may experience difficulties in other devices. To mitigate this, the program will be tested in multiple devices.