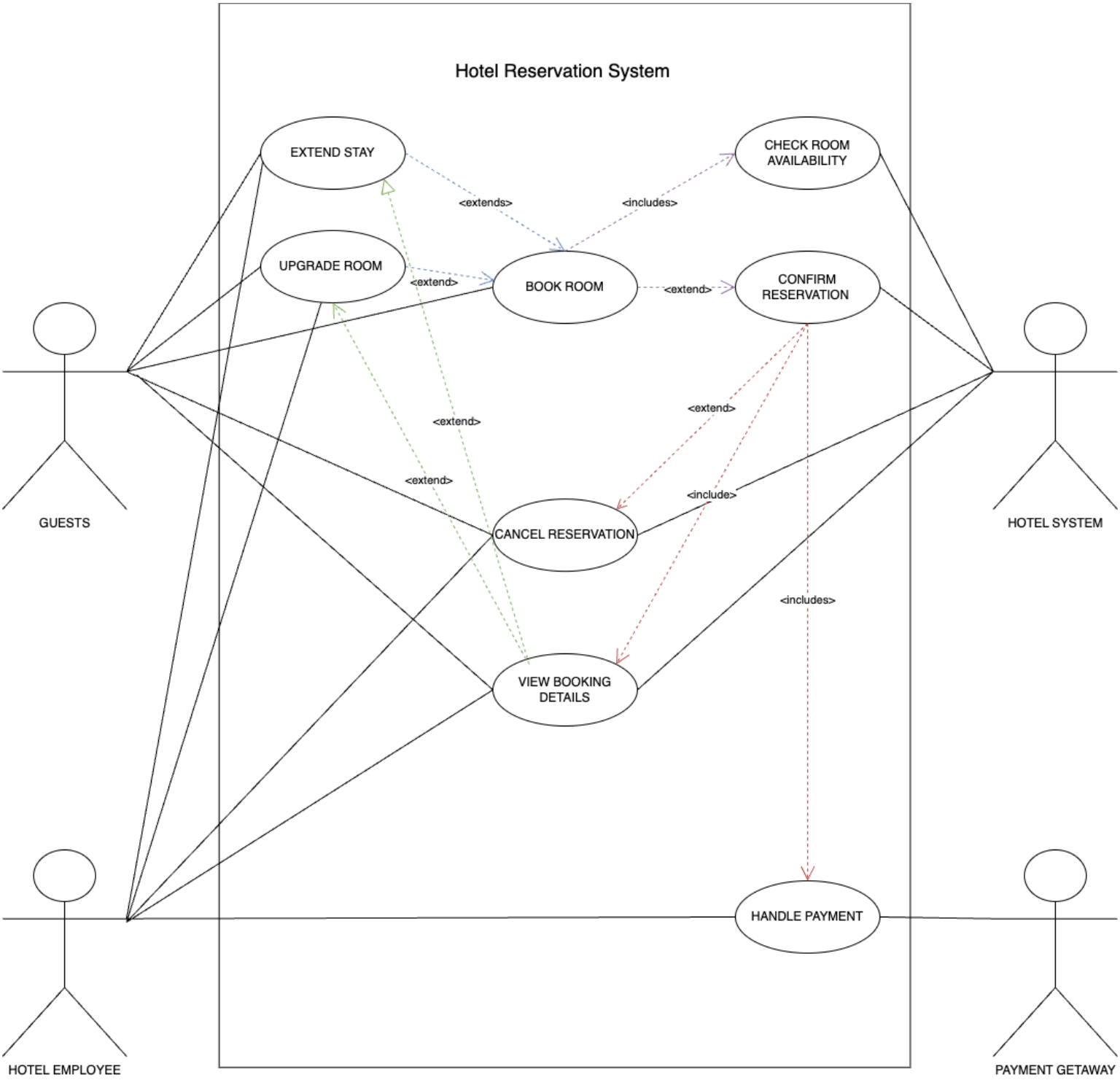
UML Use-Case Diagrams and Descriptions



This UML use case diagram outlines the hotel reservation system with four key actors: the guest, hotel employee, hotel system, and payment gateway. These actors interact with eight distinct use cases within the system boundary:

1. The first use case is Book Room, where both the guest and hotel system are involved. The guest initiates the room booking, and the hotel system checks room availability before finalizing the reservation.

2. The second use case is Cancel Reservation, where the guest, hotel employee, and hotel system collaborate. Either the guest or the hotel employee can cancel a reservation, and the hotel system updates the reservation status accordingly. The customer directly communicates with the system, while the hotel employee can assist by managing the cancellation on the guest's behalf.

3. The third use case is Upgrade Room, allowing either the guest or the hotel employee to request a room upgrade. The guest can request the upgrade directly through the system, while the hotel employee can handle the upgrade for the guest. The hotel system processes the request and updates the booking with the new room details.

4. The fourth use case is Extend Stay, which can be initiated by either the guest or the hotel employee to lengthen the duration of the stay. This use case reflects a shared relationship between both actors. The hotel system updates the reservation to reflect the extended check-out date.

5. The fifth use case, View Booking Details, enables the guest and the hotel employee to view reservation information. The hotel system facilitates this by providing access to the relevant booking details.

6. The sixth use case, Confirm Reservation, occurs after the Book Room use case. Once the room is booked, the hotel system confirms the reservation, with both the guest and the system engaged in this process.

7. The seventh use case, Handle Payment, involves the payment gateway, which processes the payment after the reservation is confirmed. This ensures that the guest completes the payment necessary for their stay.

8. Lastly, the Check Room Availability use case is directly associated with the hotel system. It checks whether a room is available during the booking process, ensuring that the system only offers available rooms for reservation.

The diagram demonstrates the relationships between use cases, showcasing <<extend>> and <<include>> connections. For instance, "Cancel Reservation" extends from "Confirm Reservation," meaning the guest or hotel employee can only cancel a reservation once it has been confirmed. Similarly, "Upgrade Room" extends from "Book Room," as upgrading the room is an optional action that can be taken after booking. The "Extend Stay" use case also extends the system, allowing both the guest and hotel employee to extend the stay while the hotel system updates the booking accordingly.

Additionally, "Handle Payment" is included in "Confirm Reservation," as payment processing is a mandatory step after confirming a reservation. "View Booking Details" extends from "Confirm Reservation," allowing guests and hotel employees to view the reservation details once it has been confirmed. Moreover, "Check Room Availability" is included in "Book Room," as checking availability is a required part of the booking process.

UML use case descriptions for each use case:

Use Case:	Book Room
Actors	Guest, Hotel System
Trigger	A guest wants to book a room through the hotel reservation system.
Preconditions	- The guest has an account and is logged into the system. - Rooms are available in the hotel for the selected dates.
Main Scenario	1. The guest searches for available rooms. 2. The guest selects a room based on preferences. 3. The guest provides payment details. 4. The system confirms the reservation and provides the booking confirmation.
Exceptions	1. If no rooms are available, the system prompts the guest to choose alternate dates. 2. If payment fails, the system prompts the guest to retry or provide alternative payment methods.

Use Case:	Cancel Reservation
Actors	Guest, Hotel Employee, Hotel System
Trigger	A guest or hotel employee wants to cancel a confirmed reservation.
Preconditions	- A reservation must already be confirmed.
Main Scenario	1. The guest or hotel employee selects the reservation to cancel. 2. The system cancels the reservation and updates the reservation status.
Exceptions	1. If the cancellation deadline has passed, the system notifies the guest or hotel employee that the reservation cannot be canceled.

Use Case:	Upgrade Room
Actors	Guest, Hotel Employee, Hotel System
Trigger	A guest or hotel employee requests a room upgrade after a booking.
Preconditions	- The guest must have a confirmed booking. - Upgraded rooms must be available.
Main Scenario	1. The guest or hotel employee requests an upgrade. 2. The system checks for available upgraded rooms. 3. The system confirms the upgrade and updates the booking.
Exceptions	1. If no upgraded rooms are available, the system notifies the guest or hotel employee of unavailability.

Use Case:	Extend Stay

Actors	Guest, Hotel Employee, Hotel System
Trigger	A guest or hotel employee requests a room upgrade after a booking.
Preconditions	- The guest must have a confirmed booking. - Availability of the room must be confirmed for the new dates.
Main Scenario	1. The guest or hotel employee requests an extension. 2. The system checks for room availability on the new dates. 3. The system updates the booking with the extended stay.
Exceptions	1. If the room is unavailable for the extended dates, the system notifies the guest or hotel employee of the unavailability.

Use Case:	View Booking Details
Actors	Guest, Hotel Employee, Hotel System
Trigger	A guest or hotel employee wants to view the details of an existing booking.
Preconditions	- A valid booking must exist in the system.
Main Scenario	1. The guest or hotel employee requests to view the booking details. 2. The system retrieves and displays the booking details.
Exceptions	1. If the booking does not exist, the system notifies the guest or hotel employee.

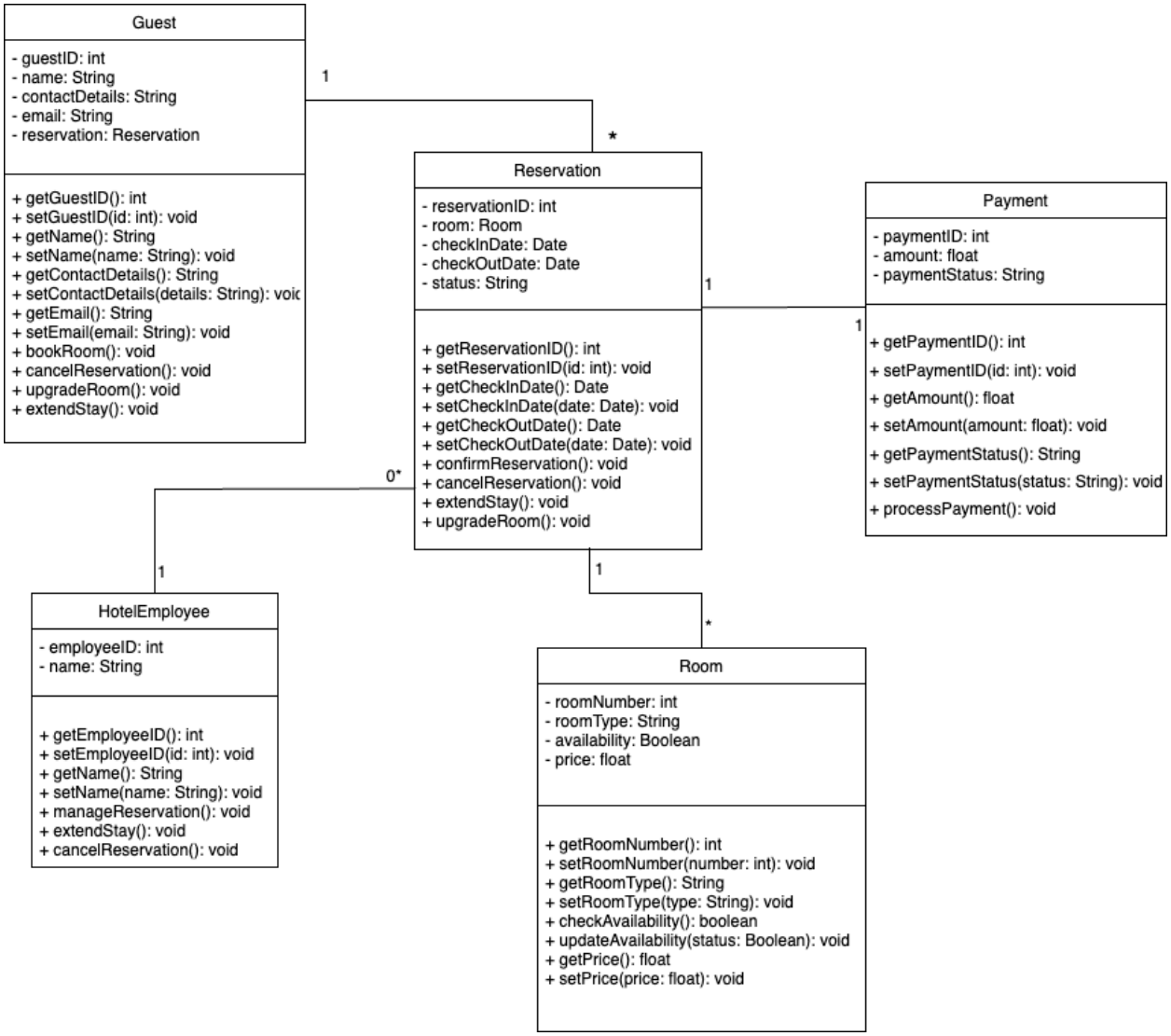
Use Case:	Confirm Reservation
Actors	Guest, Hotel System
Trigger	A room has been booked, and the system needs to confirm the reservation.
Preconditions	- A valid room must be booked.
Main Scenario	1. The system verifies the room availability. 2. The system confirms the reservation and provides a booking confirmation to the guest.

Exceptions	1. If the room is no longer available, the system notifies the guest and prompts them to choose another room or dates.

Use Case:	Handle Payment
Actors	Guest, Payment Gateway
Trigger	A guest needs to complete payment after confirming a reservation.
Preconditions	- The reservation must be confirmed. - The payment gateway must be operational.
Main Scenario	1. The guest provides payment details. 2. The payment gateway processes the payment. 3. The system confirms that payment has been received and updates the reservation status.
Exceptions	1. If payment fails, the system notifies the guest and prompts them to retry or provide alternate payment methods.

Use Case:	Check Room Availability
Actors	Hotel System
Trigger	A room booking request is initiated, and the system needs to check room availability.
Preconditions	- The system must have real-time information on room availability.
Main Scenario	1. The system checks the availability of rooms for the selected dates. 2. The system presents available rooms or notifies the guest of unavailability.
Exceptions	1. If the system cannot access real-time availability, it notifies the guest or hotel employee to try again later.

UML Class Diagrams and Description



Guest:

The attributes in this class, such as guestID, name, contactDetails, and email, describe the guest's personal data. We may state that a Guest "HAS" reservations because of the association link between the Guest and Reservation classes. As the figure illustrates, it is specifically a one-to-many connection, meaning that a "1" guest may have "*" bookings. Additionally, the program covers how to make bookings, change them, get better accommodations, and stay longer.

Reservation:

This class has attributes like reservationID, checkInDate, checkOutDate, status, and a reference to the Room object that reflect the reservation information of a hotel booking. We may state that a reservation "HAS" a room because of the association link between the Reservation and Room classes. A "1" reservation is linked to a "1" room because of the one-to-one connection. Furthermore, Payment and Reservation are associated, meaning that a reservation "HAS" one payment. There are several ways to make reservations: you may confirm, modify, cancel, and extend your stay.

Room:

Aspects about the hotel room, including the room number, kind, availability, and cost, are contained in the Room class. A reservation is linked to a room and is shown as a one-to-one relationship using the Reservation class. The session covers how to update the room's data and check availability for any of the three available room types: standard, deluxe, or suite.

Payment:

The Payment class contains details about payments made for reservations, with attributes such as paymentID, amount, and paymentStatus. A Payment "HAS" a reservation, as each payment is linked to a specific reservation. This is a one-to-one relationship where "1" payment corresponds to "1" reservation. The class includes methods to handle payment processing and update payment status.

HotelEmployee:

The HotelEmployee class includes the personal details of hotel staff, such as employeeID and name. A HotelEmployee "HAS" reservations, which is a zero-to-many relationship, meaning that a hotel employee can manage many reservations or none at all. The class has methods to manage, extend, and cancel reservations on behalf of guests.

CODE:

CODE LINK:

https://github.com/bshayerghanim9088/202203206/tree/main

```
from enum import Enum

from datetime import datetime



"""Reservation System Classes"""



# RoomType Enum for room types

class RoomType(Enum):

   Queen = 1

   King = 2

   Suite = 3



# Guest class represents the hotel guest

class Guest:

   def __init__(self, guestID=0, name="", contactDetails="", email="",
address="", phone="", reservation=None):

       self.guestID = guestID  # Unique identifier for the guest

       self.name = name  # Guest's name

       self.contactDetails = contactDetails  # Guest's contact information
```

```
        self.email = email  # Guest's email address
        self.address = address  # Guest's address
        self.phone = phone  # Guest's phone number
        self.reservation = reservation  # Reservation associated with the guest

    # Getters for guestID, name, email, contact details, address, and phone
    def getGuestID(self):
        return self.guestID

    def getName(self):
        return self.name

    def getEmail(self):
        return self.email

    def getAddress(self):
        return self.address

    def getPhone(self):
        return self.phone

# Reservation class manages the reservation process
class Reservation:
    def __init__(self, reservationID=0, room=None, checkInDate=None,
checkOutDate=None, status="", advance_payment=None):
        self.reservationID = reservationID  # Unique identifier for the
reservation
```

```
        self.room = room  # Room object associated with the reservation
        self.checkInDate = checkInDate  # Check-in date
        self.checkOutDate = checkOutDate  # Check-out date
        self.status = status  # Reservation status (e.g., Confirmed, Canceled)
        self.advance_payment = advance_payment  # Payment object linked to the
reservation


    # Getters for reservationID, check-in, and check-out dates
    def getReservationID(self):
        return self.reservationID


    def getCheckInDate(self):
        return self.checkInDate


    def getCheckOutDate(self):
        return self.checkOutDate


# Room class to represent the hotel room
class Room:
    def __init__(self, roomNumber=0, roomType=RoomType.Queen,
pricePerNight=0.0):
        self.roomNumber = roomNumber  # Room number
        self.roomType = roomType  # Type of room
        self.pricePerNight = pricePerNight  # Price per night


    # Getters for room type and price per night
    def getRoomNumber(self):
```

```
        return self.roomNumber


    def getRoomType(self):

        return self.roomType


    def getPricePerNight(self):

        return self.pricePerNight


# Payment class handles the payment processing

class Payment:

    def __init__(self, paymentID=0, amount=0.0, cardNumber="", paymentMethod="",
paymentNote=""):

        self.paymentID = paymentID  # Unique payment ID

        self.amount = amount  # Payment amount

        self.cardNumber = cardNumber  # Card number used for payment

        self.paymentMethod = paymentMethod  # Payment method (e.g.,
"MasterCard")

        self.paymentNote = paymentNote  # Payment note (e.g., "Ending in 9804")


    # Getters for masked card number and payment details

    def getMaskedCardNumber(self):

        return f"**** **** **** {self.cardNumber[-4:]}"


# Generate Invoice

def generate_invoice(guest):

    """Generate an invoice for the reservation."""

    reservation = guest.reservation
```

```
   print("Your Reservation Is Confirmed")

   print("Thank you for your reservation. Please print your hotel receipt and
show it at checkin\n")



   print(f"Your Name: {guest.getName()}")

   print(f"Your Email: {guest.getEmail()}")

   print(f"Priceline Trip Number: {reservation.advance_payment.paymentID}")

   print(f"Hotel Confirmation Number: {reservation.getReservationID()}\n")



   print("Comfort Inn & Suites Los Alamos")

   print(f"Address: {guest.getAddress()}")

   print(f"Phone: {guest.getPhone()}")

   print(f"Room Number: {reservation.room.getRoomNumber()}: {guest.getName()}")

   print(f"Check-In: {reservation.getCheckInDate()}")

   print(f"Check-Out: {reservation.getCheckOutDate()}")

   num_nights = (reservation.getCheckOutDate() -
reservation.getCheckInDate()).days

   print(f"Number of Nights: {num_nights}")

   print(f"Number of Rooms: 1")

   print(f"Room Type: {reservation.room.getRoomType().name}")



   print("\nSummary of Charges")

   print(f"Billing Name: {guest.getName()}")

   print(f"{reservation.advance_payment.paymentMethod}:
{reservation.advance_payment.paymentNote}")

   print(f"Room Cost per night: ${reservation.room.getPricePerNight():.2f}")

   print(f"Rooms: 1")
```

```
    print(f"Nights: {num_nights}")

    room_subtotal = reservation.room.getPricePerNight() * num_nights

    print(f"Room Subtotal: ${room_subtotal:.2f}")

    taxes_and_fees = 21.58  # Example value for taxes and fees

    print(f"Taxes and Fees: ${taxes_and_fees:.2f}")

    total_amount = room_subtotal + taxes_and_fees

    print(f"Total Charges: ${total_amount:.2f}")



# Sample Data to Create Objects and Generate Invoice



# Create Room object

room1 = Room(roomNumber=101, roomType=RoomType.Queen, pricePerNight=89.85)



# Create Payment object

payment1 = Payment(paymentID=52573887, amount=201.48,
cardNumber="1234567890129804", paymentMethod="MasterCard", paymentNote="Ending
in 9804")



# Create Reservation object

check_in_date = datetime(2024, 10, 10)

check_out_date = datetime(2024, 10, 15)

reservation1 = Reservation(reservationID=123456, room=room1,
checkInDate=check_in_date, checkOutDate=check_out_date, status="Confirmed",
advance_payment=payment1)



# Create Guest object

guest1 = Guest(guestID=1001, name="Ali Al-Hashmi", contactDetails="0567891234",
email="ali.alhashmi@example.com", address="Abu Dhabi", phone="0567891234",
reservation=reservation1)
```

```
# Generate and display the invoice

generate_invoice(guest1)
```

Summary of learnings:

I went through a number of crucial phases in order to finish developing the hotel reservation system for this task. I started by studying how to create UML use case diagrams, which made it easier for me to comprehend how different actors and use cases interacted. I specifically made a use case diagram for the hotel reservation system that illustrates the interactions between the guest, the hotel system, hotel staff, and the payment gateway in order to fulfill different use cases, such as making a reservation or processing a payment. I also carefully considered the connections between these use cases, selecting the appropriate instances of "include" and "extend" to indicate required or optional activities. I then proceeded to write thorough use case descriptions for every use case, including information on the actors, preconditions, triggers, key situations, and any exceptions. I was able to sketch out each interaction's flow throughout the system with clarity thanks to this phase. After that, I created UML class diagrams for each of the system's primary classes, including HotelEmployee, Guest, Reservation, Room, and Payment. I determined the proper relationships whether associations or one-to-many relationships—between these classes throughout this step. Based on each class's functions inside the system, I also gave much thought to the characteristics and approaches that would be appropriate for them. At last, I converted the UML diagrams into working code. I applied the necessary logic and procedures, making use of every programming idea we had discussed in earlier sessions. For each class, I made instances (objects) to mimic the operation of an actual reservation system. By using these techniques, for instance, I was able to display the room kinds, payment data, and reservation details, giving the system tangible functionality. Through this method, I was able to use object-oriented ideas in a real-world setting and strengthen my understanding of them.