Hotel Reservation system Design

Name

StudentID

University

# Introduction

This project concentrates mainly on creating and rolling out a software system that oversees hotel reservations in a hotel in Abu Dhabi. This system intends to simplify reservations, increase customer satisfaction, and facilitate the smooth operation of payment and booking. The system design permits customers to either reserve online or directly at the hotel, with structured confirmation provided. The project looks at the entire hotel reservation lifecycle, from evaluating room availability to processing payment and dealing with cancellations. The software system introduces numerous UML diagrams, including use-case and class diagrams, to deliver a complete view of system architecture and behavior. The core classes that include the attributes and behaviors necessary for effective hotel reservation management use Python programming.

# UML Case diagram

The UML class diagram for the hotel reservation system is built to illustrate the central components of the system along with their relationships, giving a transparent view of how different objects behave and encapsulate data in the system.

Reservation data related to the customer is stored in the Customer class. This class is generally comprised of attributes that include the customer's name, contact information, and hopefully an identification number. Reservations or payments cannot be initiated unless the customer participates.

The hotel class represents the organization from which the reservations originate. It includes information about the hotel's label, location, and contact information. This class is essential because it connects to the rooms available for reservation and provides the context for those reservations.

Room class models depict the individual accommodations provided by the hotel. They can encompass room number, kind of room, availability status, and pricing information. The functionality of the room class is critical, as it is bonded directly to both the customer and reservation processes. It assures the system can effectively manage inventory by tracking the status of rooms, indicating whether they are occupied, in use, or needing attention.

The Reservation class serves as the system's basis, enclosing all the information about bookings made by each customer. This class will find attributes, including the reservation number, the dates of check-in and check-out, and the number of nights. Intensely associated with the room and customer classes, it must unite the customer with their reserved room and monitor room availability.

The Payment class handles all the financial transactions associated with reservations. Documenting the payment method and amount, along with taxes, the transaction handles the fees tied to it. This class makes certain that the appropriate fees are applied during the reservation process for making, changing, or canceling a reservation. It also keeps control of the status of every payment, whether it's fulfilled, pending, or refunded.

These classes illustrate how data and behavioral dynamics interact throughout the system. For example, a customer might have several reservations, each associated with a separate room and using the payment system. In addition to clarifying the system's static framework, the diagram acts as a template for the dynamic exchanges that happen during the reservation lifecycle.

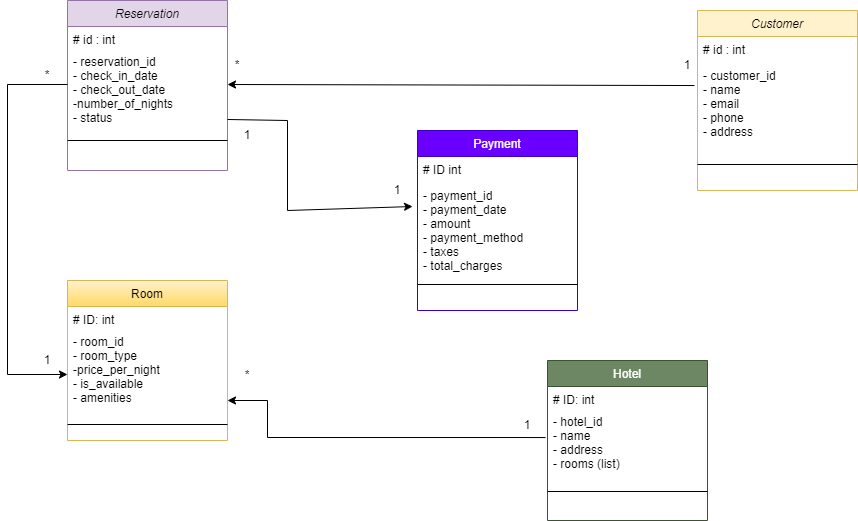


Figure 1:UML Diagram

# Use case diagram

The UML use-case diagram for the hotel reservation system outlines the primary functionalities the system must support, specifically focusing on three core scenarios:

## Reservation Process:

The process enables the customer to schedule a room reservation. The framework encompasses use cases that include "Make a Reservation," "Check Room Availability," and "Generate Reservation Confirmation." These scenarios guarantee that customers can spot available rooms, choose suitable dates, and get a confirmation after their reservation procedure has concluded. In this instance, the relationship between use cases is important because the system has to 'include' room availability before generating confirmation.

## Payment Handling:

Within this scenario, we address the financial considerations of the reservation system. After confirming their booking, the customer has to pay, facilitated through the use cases "Process Payment" and "Apply Taxes and Fees." These measures ensure customers are charged the right amount, inclusive of taxes and any additional fees. These use scenarios are connected since the processing of payments automatically results in adding taxes.

## Reservation Management:

The reservation framework also facilitates the modification and cancellation of currently booked reservations. Use cases, including 'Modify Reservation,' 'Cancel Reservation,' and 'Process Refund,' handle these changes. The system ensures that customers can adjust or cancel their bookings if necessary while also processing refunds where warranted. The 'extend' relationship applies here because refund processing is needed solely when a reservation is canceled.

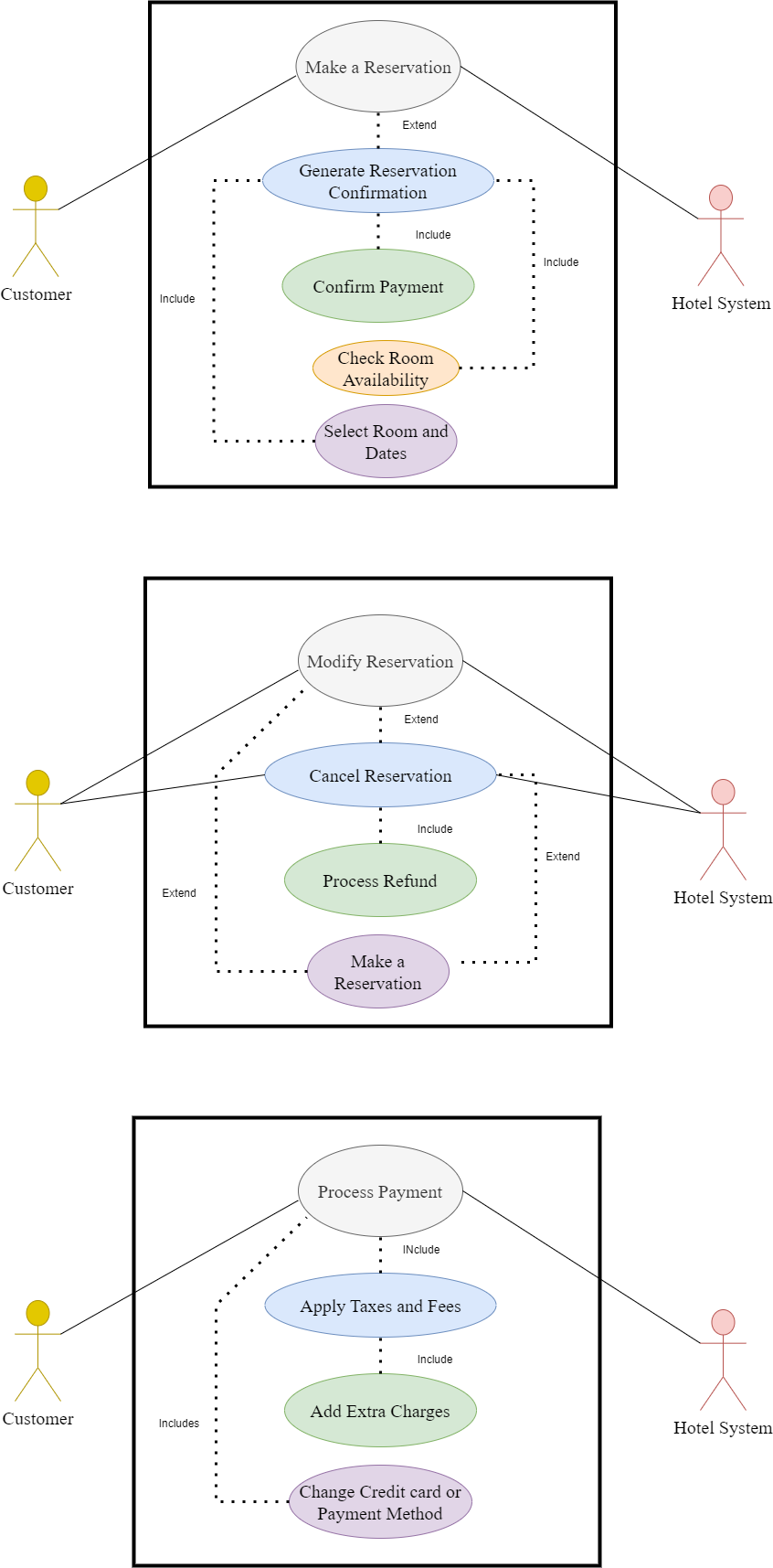


Figure 2: Use Case Diagram

# Python Code

**HotelReservationSytem.py:**

from datetime import datetime

from Customer import Customer

from Room import Room

from Reservation import Reservation

from Payment import Payment

class HotelReservationSystem:

def \_\_init\_\_(self):

# Sample data

self.customers = []

self.rooms = []

self.reservations = []

def add\_customer(self, name, email, phone):

customer = Customer(name, email, phone)

self.customers.append(customer)

return customer

def add\_room(self, room\_number, room\_type, price\_per\_night):

room = Room(room\_number, room\_type, price\_per\_night)

self.rooms.append(room)

return room

def make\_reservation(self, customer, room, check\_in, check\_out):

if room.book\_room():

reservation = Reservation(customer, room, check\_in, check\_out)

self.reservations.append(reservation)

payment = Payment(reservation, "Credit Card")

payment.process\_payment()

print(reservation.get\_reservation\_details())

else:

print("Room is not available!")

# Testing the implementation

if \_\_name\_\_ == "\_\_main\_\_":

system = HotelReservationSystem()

# Adding a customer

customer1 = system.add\_customer("Ted Vera", "tedvera@mac.com", "555-1234")

# Adding a room

room1 = system.add\_room(101, "Double", 89.95)

# Making a reservation

check\_in\_date = datetime(2023, 8, 22)

check\_out\_date = datetime(2023, 8, 24)

system.make\_reservation(customer1, room1, check\_in\_date, check\_out\_date)

**Customer.py:**

class Customer:

def \_\_init\_\_(self, name, email, phone):

self.name = name # Customer's name

self.email = email # Customer's email

self.phone = phone # Customer's phone number

def get\_details(self):

return f"Name: {self.name}, Email: {self.email}, Phone: {self.phone}"

def update\_email(self, new\_email):

self.email = new\_email # Update the customer's email

def update\_phone(self, new\_phone):

self.phone = new\_phone # Update the customer's phone number

**Payment.py**

class Payment:

def \_\_init\_\_(self, reservation, payment\_method):

self.reservation = reservation # Instance of Reservation

self.payment\_method = payment\_method # Payment method (e.g., credit card)

def process\_payment(self):

# Placeholder for payment processing logic

print(f"Processing payment of {self.reservation.total\_cost} via {self.payment\_method}")

return True # Assume payment is successful

def get\_payment\_details(self):

return (f"Payment Details:\n"

f"Reservation Cost: {self.reservation.total\_cost}\n"

f"Payment Method: {self.payment\_method}")

**Reservation.py:**

class Reservation:

def \_\_init\_\_(self, customer, room, check\_in\_date, check\_out\_date):

self.customer = customer # Instance of Customer

self.room = room # Instance of Room

self.check\_in\_date = check\_in\_date # Check-in date

self.check\_out\_date = check\_out\_date # Check-out date

self.total\_cost = self.calculate\_cost() # Calculate total cost upon reservation

def calculate\_cost(self):

nights = (self.check\_out\_date - self.check\_in\_date).days # Calculate number of nights

return nights \* self.room.price\_per\_night # Total cost

def get\_reservation\_details(self):

return (f"Reservation Details:\n"

f"Customer: {self.customer.get\_details()}\n"

f"Room: {self.room.get\_room\_details()}\n"

f"Check-in: {self.check\_in\_date}\n"

f"Check-out: {self.check\_out\_date}\n"

f"Total Cost: {self.total\_cost}")

**Room.py:**

class Room:

def \_\_init\_\_(self, room\_number, room\_type, price\_per\_night):

self.room\_number = room\_number # Room number

self.room\_type = room\_type # Type of the room (e.g., single, double)

self.price\_per\_night = price\_per\_night # Price per night for the room

self.is\_available = True # Room availability status

def book\_room(self):

if self.is\_available:

self.is\_available = False # Mark the room as booked

return True

return False

def release\_room(self):

self.is\_available = True # Mark the room as available again

def get\_room\_details(self):

return f"Room Number: {self.room\_number}, Type: {self.room\_type}, Price per Night: {self.price\_per\_night}, Available: {self.is\_available}"

# Github link

# Summary of learning

While developing the hotel reservation system, crucial learning milestones emerged in system design and software development. UML diagrams, mainly use-case and class diagrams, make sense of the system's structure and functionality. Our ability to break down the requirements into different use cases helped us achieve a systematic approach to the interactions between the actors and the system, leading to a more precise understanding of how the hotel reservation process operates from the perspectives of both users and hotel staff. The growth of the UML class diagram also amplified the understanding of principles related to object-oriented design, notably the understanding of key entities, including customers, rooms, reservations, hotels, and payments.

Understanding the skill to model these objects, their attributes, and their relationships has allowed insight into the data flow in the system and the interactions needed between these objects to furnish a comprehensive user experience. They were creating these UML designs with Python to improve comprehension of the concepts related to encapsulation, inheritance, and modularity. While developing Python classes with proper attributes and methods, our knowledge of class design and object-oriented programming progressed. The production of uncomplicated and functional code and thorough documentation of its features also emphasized the value of coding standards and sufficient error responses.

The project highlighted the need for system scalability and a modular design. Developing the system in established class structures and use cases supports future deployments, covering added payment options and enhanced functionality for room management without impairing the core system performance. This assignment offered us a substantial learning chance that combined theoretical design with practical coding, providing us with important skills in both software engineering and system architecture.