**Design and Implementation of a UML-Based Hotel Management System in Python**

Ahmed Abdelaziz Al Qahtani

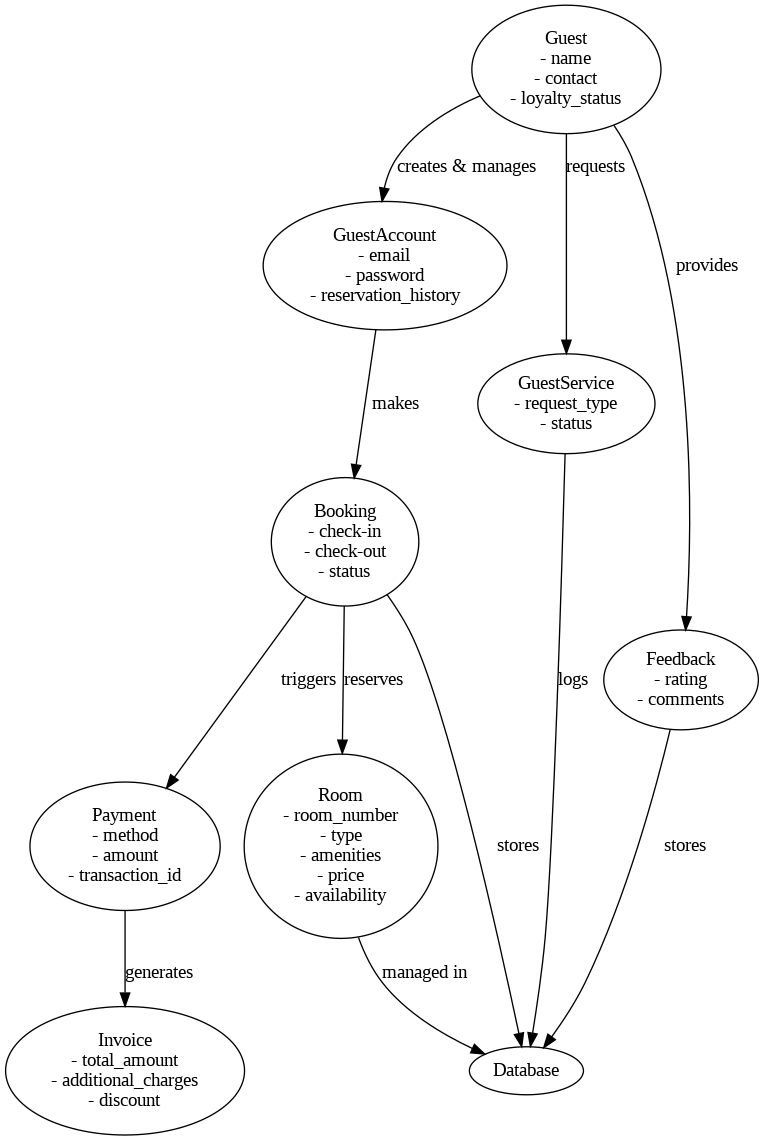
student id 202206351

Program Fund

18/03/2025

Design and Implementation of a UML-Based Hotel Management System in Python

**UML Class Diagram**

****

The Royal Stay Hotel Management System is designed using Object-Oriented Programming (OOP) principles, ensuring modularity and scalability. The key classes are:

**Guest** - Stores guest details such as name, and contact.

**GuestAccount** - Manages guest login and reservation history.

**Room -** Contains room attributes like number, type, price, and availability.

**Booking** - Handles reservation details, including dates and status.

**Payment** - Processes transactions for instance method, amount, and reference number.

**Invoice -** Generates billing details for bookings.

**Relationships**

**Composition** - GuestAccount owns multiple Booking objects.

**Association** - Booking is linked to both Guest and Room.

**Aggregation** - Invoice aggregates Booking details.

The Royal Stay Hotel Management System allowed me to further learn the Object-Oriented Analysis and Design (OOAD) principles. I structured the system with Unified Modeling Language (UML) to represent real-world hotel operations. The UML class diagram represented the interaction between different system components as a part of the critical step for efficient design without coding.

As for the Python implementation, it was built with modularity in mind, making each class a separate file. I used OOP concepts of inheritance, encapsulation, private attributes, getters, setters, and composition for maintainability. The method \_\_str\_\_() improved readability for the class, in addition, structured doc strings made it clear. The project was completely dependent on testing. I wrote code that uses unit testing and validated functionalities such as guest account creation, room reservations, invoice generation, payment processing, and booking cancellations. Writing test cases was also useful in locating potential errors as early as possible, resulting in a robust, and free of error, system.

Version control with GitHub also helped in project management because it allows for incremental progress to be tracked as well. Sharing the repository encourages collaboration and together, we can improve further. This project helped me gain a better grasp of correctly designing software so that one can use OOP best practices, and have the ability to create real world applications using modular code and structured documentation. The experience helped in bridging the gap between theoretical UML concepts and how they can be practically implemented using Python.

**Guest Code**

class Guest:

"""Represents a guest in the hotel system."""

def \_\_init\_\_(self, name: str, contact: str, loyaltyStatus: str = "Regular"):

self.\_\_name = name

self.\_\_contact = contact

self.\_\_loyaltyStatus = loyaltyStatus

# Getters

def getName(self):

return self.\_\_name

def getContact(self):

return self.\_\_contact

def getLoyaltyStatus(self):

return self.\_\_loyaltyStatus

# Setters

def setContact(self, newContact):

self.\_\_contact = newContact

def setLoyaltyStatus(self, newStatus):

self.\_\_loyaltyStatus = newStatus

def \_\_str\_\_(self):

return f"Guest: {self.\_\_name}, Contact: {self.\_\_contact}, Loyalty: {self.\_\_loyaltyStatus}"

**Guest Account Code**

class GuestAccount:

"""Handles guest account login and profile updates."""

def \_\_init\_\_(self, email: str, password: str):

self.\_\_email = email

self.\_\_password = password

self.\_\_reservationHistory = []

# Getters

def getEmail(self):

return self.\_\_email

def getReservationHistory(self):

return self.\_\_reservationHistory

# Methods

def addReservation(self, reservation):

self.\_\_reservationHistory.append(reservation)

def \_\_str\_\_(self):

return f"GuestAccount: {self.\_\_email}, Reservations: {len(self.\_\_reservationHistory)}"

**Guest Service Code**

class GuestService:

"""Handles guest service requests."""

def \_\_init\_\_(self, requestType: str):

self.\_\_requestType = requestType

self.\_\_status = "Pending"

def completeRequest(self):

self.\_\_status = "Completed"

def \_\_str\_\_(self):

return f"Service Request: {self.\_\_requestType} - {self.\_\_status}"

**Invoice Code**

class Invoice:

"""Represents an invoice for a booking."""

def \_\_init\_\_(self, booking, additionalCharges=0, discount=0):

self.\_\_booking = booking

self.\_\_totalAmount = booking.\_Booking\_\_room.getPrice() + additionalCharges - discount

def \_\_str\_\_(self):

return f"Invoice: {self.\_\_totalAmount} (Booking: {self.\_\_booking})"

**Payment Code**

class Payment:

"""Handles payment transactions."""

def \_\_init\_\_(self, method: str, amount: float, transactionId: str):

self.\_\_method = method

self.\_\_amount = amount

self.\_\_transactionId = transactionId

def \_\_str\_\_(self):

return f"Payment: {self.\_\_method} | Amount: ${self.\_\_amount} | Transaction ID: {self.\_\_transactionId}"

**Feedback Code**

class Feedback:

"""Stores guest feedback and ratings."""

def \_\_init\_\_(self, guest, rating: int, comments: str):

self.\_\_guest = guest

self.\_\_rating = rating

self.\_\_comments = comments

def \_\_str\_\_(self):

return f"Feedback from {self.\_\_guest.getName()}: {self.\_\_rating}/5 - {self.\_\_comments}"

**Room Code**

class Room:

"""Represents a hotel room."""

def \_\_init\_\_(self, roomNumber: int, roomType: str, amenities: list, price: float):

self.\_\_roomNumber = roomNumber

self.\_\_roomType = roomType

self.\_\_amenities = amenities

self.\_\_price = price

self.\_\_available = True # Initially available

# Getters

def getRoomNumber(self):

return self.\_\_roomNumber

def getRoomType(self):

return self.\_\_roomType

def isAvailable(self):

return self.\_\_available

# Methods

def bookRoom(self):

"""Marks the room as unavailable."""

self.\_\_available = False

def releaseRoom(self):

"""Marks the room as available again."""

self.\_\_available = True

def \_\_str\_\_(self):

status = "Available" if self.\_\_available else "Booked"

return f"Room {self.\_\_roomNumber}: {self.\_\_roomType} - {status}"

**Booking Code**

from datetime import datetime

class Booking:

"""Handles guest room reservations."""

def \_\_init\_\_(self, guest, room, checkIn: str, checkOut: str):

self.\_\_guest = guest

self.\_\_room = room

self.\_\_checkIn = datetime.strptime(checkIn, "%Y-%m-%d")

self.\_\_checkOut = datetime.strptime(checkOut, "%Y-%m-%d")

self.\_\_status = "Confirmed"

# Methods

def cancelBooking(self):

"""Cancels the booking and releases the room."""

self.\_\_status = "Cancelled"

self.\_\_room.releaseRoom()

def \_\_str\_\_(self):

return f"Booking for {self.\_\_guest.getName()} in Room {self.\_\_room.getRoomNumber()} ({self.\_\_status})"

**Main Code**

from Guest import Guest

from Room import Room

from Booking import Booking

from Payment import Payment

from GuestService import GuestService

from Feedback import Feedback

# Creating a guest

guest1 = Guest("James Alaine", "+1 787-456-823")

# Creating a room

room1 = Room(101, "Deluxe", ["WiFi", "TV", "Mini-Bar"], 150)

# Making a booking

booking1 = Booking(guest1, room1, "2025-03-10", "2025-03-15")

print(booking1)

# Processing payment

payment1 = Payment("Credit Card", 750, "TXN12345")

print(payment1)

# Guest leaves feedback

feedback1 = Feedback(guest1, 5, "Excellent service!")

print(feedback1)

Github link: <https://github.com/ByteKhalifa/royalhotel.git>