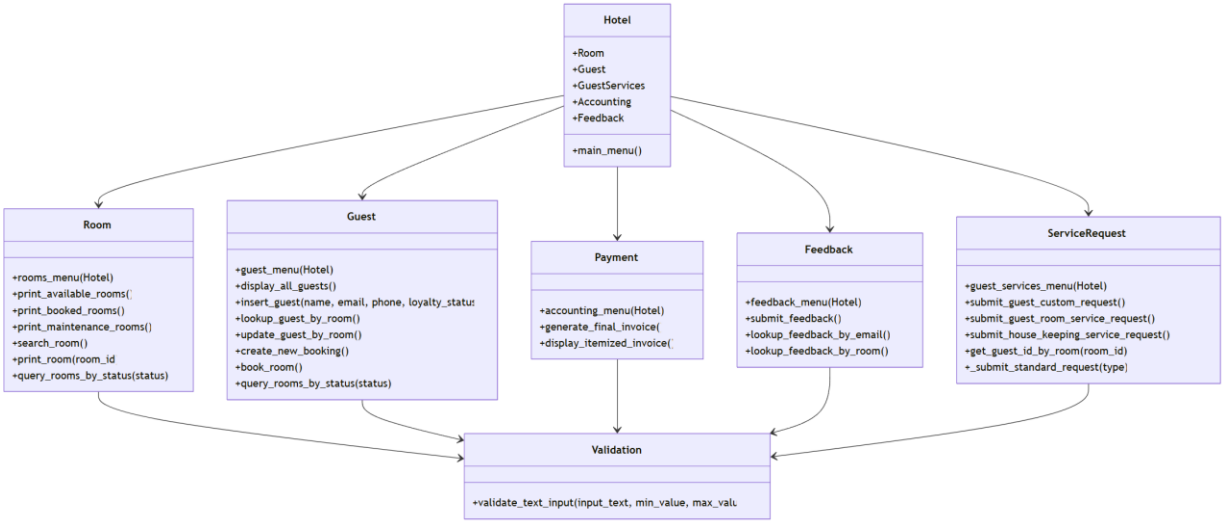
Royal Stay Hotel Management System

Abdulrahman Alzaabi

ICS220 Program. Fund.

Areej Abdulfattah

A. UML Case Diagram



B. Description

The system is represented in a UML class diagram with the following class structure and relationships:

Classes:

- **Hotel**: Central controller; aggregates and coordinates all other modules.
- **Room**: Manages room queries, availability, and display.
- **Guest**: Handles guest creation, lookup, updates, and bookings.
- **Payment**: Manages invoice generation and payment processing.
- **ServiceRequest**: Handles service request submission and tracking.
- **Feedback**: Manages feedback submission and lookup.
- **Validation**: Utility class for input validation used by all modules.

Relationships:

- Hotel **aggregates** Room, Guest, Payment, ServiceRequest, and Feedback.
- Each of Room, Guest, Payment, ServiceRequest, and Feedback **uses** Validation.
- Methods within each module represent functional operations related to hotel workflows.

This structure ensures separation of concerns and easy maintenance, extension, and testing.

C. Project Structure

```
├── main.py              # Entry point for the application
├── hotelSystem.py       # Central controller coordinating all modules
├── roomManagement.py    # Handles room availability and details
├── guestManagement.py   # Manages guest bookings and guest records
├── paymentInvoicing.py  # Handles invoicing and accounting
├── guestServices.py     # Submits and tracks service requests
├── feedbackReviews.py   # Allows guests to leave feedback and ratings
├── inputValidation.py   # Utility module for validating user input
├── dataHandler.py       # (Utility) Used to populate database and define schema
└── data.sqlite          # SQLite database for persistent storage
```

D. Core Concepts

- **Modularity**: Each major functionality (e.g., Rooms, Guests, Payments) is encapsulated in its own class/module.
- **SQLite Integration**: Data is stored persistently in data.sqlite, allowing for a lightweight yet powerful database.
- **Tabulated Display**: tabulate is used to format output cleanly in the terminal.
- **Input Validation**: Centralized in inputValidation.py to ensure robust user input handling.

E. Application Flow

main.py

- Connects to the database
- Initializes all modules
- Displays an ASCII welcome banner
- Launches the main menu via Hotel.main_menu()

hotelSystem.py

- Acts as the controller that links all subsystem modules
- Routes main menu input to the corresponding module

F. Module Overviews

roomManagement.py

- View available, occupied, or maintenance rooms
- Search room by ID
- Uses status filtering to fetch appropriate records

guestManagement.py

- Create new bookings (adds guest if not present)
- List all guests and lookup by room
- Update guest records
- Automatically handles loyalty statuses (Bronze → Silver → Gold)

paymentInvoicing.py

- Generate final invoice (including stay and service fees)
- Itemized charge breakdown
- Confirms credit card payment and records it in ACCOUNTING

guestServices.py

- Request types: Housekeeping, Room Service, Custom
- Assigns request fees
- Links request to current guest based on room

feedbackReviews.py

- Allows guests to submit ratings and comments
- Look up feedback by guest email or room number

inputValidation.py

- Validates if input is a number within a given range
- Used across all modules for menu navigation and form inputs

dataHandler.py

- (Optional utility script)
- Defines the SQLite schema and inserts dummy requests for testing

G. Input Validation Script

The system uses a centralized input validation utility in inputValidation.py to ensure all numerical user inputs are safe and within expected ranges.

Test Case Script:

```
rom inputValidation import Validation

# Test cases for the validate_text_input function

test_cases = [

    ("abc", 1, 5),         # Not a number

    ("4.5", 1, 5),         # Not a whole number

    ("-1", 1, 5),          # Out of lower bound

    ("10", 1, 5),          # Out of upper bound

    ("2", 1, 5),           # Valid input

    ("0", 1, 5),           # Boundary test - just below range

    ("1", 1, 5),           # Boundary test - lower limit

    ("5", 1, 5),           # Boundary test - upper limit

    ("3.0", 1, 5),         # Float but whole number

    ("   3   ", 1, 5),     # Input with whitespace

]

print("\nRunning Validation Test Cases:\n")

for idx, (input_val, min_val, max_val) in enumerate(test_cases, 1):

    status, result = Validation.validate_text_input(input_val.strip(), min_val, max_val)

    print(f"Test Case {idx}: Input='{input_val}' | Range=({min_val}-{max_val})")

    print(f"Result: {'Valid' if status else 'Invalid'} - {result}\n")
```

Test Case Results:

Running Validation Test Cases:

Test Case 1: Input='abc' | Range=(1-5)
Result: Invalid - Input is not a number.

Test Case 2: Input='4.5' | Range=(1-5)
Result: Invalid - Number is not a whole number.

Test Case 3: Input='-1' | Range=(1-5)
Result: Invalid - Number is not within the range 1 to 5.

Test Case 4: Input='10' | Range=(1-5)
Result: Invalid - Number is not within the range 1 to 5.

Test Case 5: Input='2' | Range=(1-5)
Result: Valid - 2

Test Case 6: Input='0' | Range=(1-5)
Result: Invalid - Number is not within the range 1 to 5.

Test Case 7: Input='1' | Range=(1-5)
Result: Valid - 1

Test Case 8: Input='5' | Range=(1-5)
Result: Valid - 5

Test Case 9: Input='3.0' | Range=(1-5)
Result: Valid - 3

Test Case 10: Input=' 3 ' | Range=(1-5)
Result: Valid - 3

This approach prevents runtime crashes and provides user-friendly error messages.

H.
I. Technologies Used

- **Python 3.10+**
- **SQLite3**: Embedded database
- **Tabulate**: For pretty-printing tables

J. How to Run

python main.py

Make sure you have Python installed and tabulate available:

pip install tabulate

K. Code

```
import sqlite3
from roomManagement import Room
from guestManagement import Guest
from paymentInvoicing import Payment
from guestServices import ServiceRequest as GuestServices
from feedbackReviews import Feedback
from hotelSystem import Hotel

# Database file name
db_name = 'data.sqlite'

# Connect to the SQLite database (it will be created if it doesn't exist)
conn = sqlite3.connect(db_name)
cursor = conn.cursor()

# Initialize modules
Room = Room(conn=conn)
Guest = Guest(conn=conn)
Payment = Payment(conn=conn)
GuestServices = GuestServices(conn=conn)
Feedback = Feedback(conn=conn)
Hotel = Hotel(Room=Room, Guest=Guest, GuestServices=GuestServices,
Accounting=Payment, Feedback=Feedback)

if __name__ == "__main__":
    ascii_art = "    ___   ___  ___   _____ _______    ________    ___   _________  _________  ___
___      \n   |\\  \\ |\\  \\|\\  \\ |\\   _ \\  _   \\ |\\  ___ \\  |\\  \\ |\\   __  \\ |\\   __  \\ |\\  \\|\\  \\    \\n   \\ \\  \\\\
\\  \\\\\\  \\\\ \\  \\\\\\__\\ \\  \\\\ \\   __/| \\ \\  \\\\ \\  \\|\\  \\\\ \\  \\|\\  \\\\ \\  \\\\\\  \\   \\n __ \\ \\  \\\\ \\  \\\\\\
\\\\ \\  \\\\|__| \\  \\\\ \\  \\_|/__\\ \\  \\\\ \\   _  _\\\\ \\   __  \\\\ \\   __  \\  \\n|\\  \\\\_\\  \\\\ \\  \\\\\\  \\\\ \\  \\
\\ \\  \\\\ \\  \\_|\\ \\\\ \\  \\\\ \\  \\\\  \\|\\ \\  \\ \\  \\\\ \\  \\ \\  \\ \\ \\n\\ \\_________\\\\ \\_________\\\\ \\__\\\\   \\
\\__\\\\ \\_________\\\\ \\__\\\\ \\__\\\\ _\\ \\ \\__\\ \\__\\\\ \\__\\ \\__\\\n \\|_________| \\|_________| \\|__|
\\|__| \\|_________| \\|___| \\|___|\\|___| \\|___|\\|___| \\|___|\\|___|\n ___  ___  _________  _________
_________  ___                                  \n|\\  \\|\\  \\ |\\   __  \\ |\\___   ___\\|\\  ___ \\  |\\  \\
\n\\ \\  \\\\\\  \\\\ \\  \\|\\  \\\\|___ \\  \\_|\\ \\   __/| \\ \\  \\                          \n \\ \\   __  \\\\ \\  \\\\\\
```

```
\\    \\ \\  \\  \\ \\  \\_|/__\\ \\  \\                          \n  \\ \\  \\ \\  \\\\ \\  \\\\\  \\    \\ \\  \\  \\ \\  \\_|\\
\\\\ \\  \\_____                        \n   \\ \\__\\ \\__\\\\ \\_________\\    \\ \\__\\\\  \\ \\_________\\\\
\\_________\\                        \n    \\|___|\\|___| \\|_________|     \\|___|   \\|_________| \\|_________|
\n"
    print(ascii_art)
    print("Jumeirah Hotel by Abood")
    print("Best Hotel Ever, Becasue we say so.")

    Hotel.main_menu()



from typing import List, Tuple
from inputValidation import Validation
from tabulate import tabulate


class Feedback:
    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn

    def feedback_menu(self, Hotel):
        """Displays the feedback menu and routes user selection."""
        print("\n\033[0;0mAvailable options:\n"
              "\033[95m1.\033[0;0m Submit Feedback\n"
              "\033[95m2.\033[0;0m Lookup Feedback by Email\n"
              "\033[95m3.\033[0;0m Lookup Feedback by Room\n"
              "\033[95m4.\033[0;0m Main Menu\n")
        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1,
max_value=4)

        if not status:
            print(value)
            return self.feedback_menu(Hotel)

        actions = {
            1: self.submit_feedback,
            2: self.lookup_feedback_by_email,
            3: self.lookup_feedback_by_room,
            4: Hotel.main_menu
        }

        action = actions.get(value)
        if action:
```

```
        action() if value == 4 else (action(), self.feedback_menu(Hotel))

    def submit_feedback(self):
        """Allows a guest to submit feedback about a room or stay."""
        print("\n--- Submit Feedback ---")
        room_number = input("Enter room number: ").strip()
        try:
            room_id = int(room_number)
            self.cursor.execute('''
                SELECT G.GUEST_ID, G.NAME FROM BOOKINGS B
                JOIN GUESTS G ON B.GUEST_ID = G.GUEST_ID
                WHERE B.ROOM_ID = ?
                ORDER BY B.CHECK_IN_DATE DESC
                LIMIT 1
            ''', (room_id,))
            result = self.cursor.fetchone()

            if not result:
                print("No guest currently associated with that room.")
                return

            guest_id, guest_name = result
            print(f"Guest: {guest_name}")
            rating_input = input("Rate your stay (1-5): ").strip()
            try:
                rating = int(rating_input)
                if not 1 <= rating <= 5:
                    raise ValueError
            except ValueError:
                print("Invalid rating. Must be an integer between 1 and 5.")
                return

            comments = input("Leave your comments: ").strip()

            self.cursor.execute('''
                INSERT INTO FEEDBACK (GUEST_ID, ROOM_ID, RATING, COMMENTS)
                VALUES (?, ?, ?, ?)
            ''', (guest_id, room_id, rating, comments))
            self.conn.commit()
            print("Thank you for your feedback!")

        except ValueError:
            print("Invalid room number entered.")

    def lookup_feedback_by_email(self):
        """Fetches and displays all feedback submitted by a guest via email."""
```

```
        print("\n--- Lookup Feedback by Email ---")
        email = input("Enter guest email address: ").strip().lower()
        self.cursor.execute('''
            SELECT G.NAME, G.EMAIL, F.ROOM_ID, F.RATING, F.COMMENTS,
F.SUBMITTED_AT
            FROM FEEDBACK F
            JOIN GUESTS G ON F.GUEST_ID = G.GUEST_ID
            WHERE G.EMAIL = ?
            ORDER BY F.SUBMITTED_AT DESC
        ''', (email,))
        feedback_entries = self.cursor.fetchall()

        if feedback_entries:
            headers = ["Name", "Email", "Room ID", "Rating", "Comments", "Submitted At"]
            print(tabulate(feedback_entries, headers=headers, tablefmt="grid"))
        else:
            print("No feedback found for that email.")

    def lookup_feedback_by_room(self):
        """Fetches and displays all feedback associated with a room."""
        print("\n--- Lookup Feedback by Room Number ---")
        room_number = input("Enter room number: ").strip()
        try:
            room_id = int(room_number)
            self.cursor.execute('''
                SELECT G.NAME, G.EMAIL, F.ROOM_ID, F.RATING, F.COMMENTS,
F.SUBMITTED_AT
                FROM FEEDBACK F
                JOIN GUESTS G ON F.GUEST_ID = G.GUEST_ID
                WHERE F.ROOM_ID = ?
                ORDER BY F.SUBMITTED_AT DESC
            ''', (room_id,))
            feedback_entries = self.cursor.fetchall()

            if feedback_entries:
                headers = ["Name", "Email", "Room ID", "Rating", "Comments", "Submitted At"]
                print(tabulate(feedback_entries, headers=headers, tablefmt="grid"))
            else:
                print("No feedback found for that room number.")
        except ValueError:
            print("Invalid room number entered.")



from inputValidation import Validation
```

```
from tabulate import tabulate

from typing import List, Tuple



class Guest:

    def __init__(self, conn):

        self.cursor = conn.cursor()

        self.conn = conn


    def guest_menu(self, Hotel):

        """Displays the guest management menu and handles user input."""

        print("\n\033[0;0mAvailable options:\n"

              "\033[95m1.\033[0;0m New Booking\n"

              "\033[95m2.\033[0;0m List all Guests\n"

              "\033[95m3.\033[0;0m Find Guest\n"

              "\033[95m4.\033[0;0m Update Guest Information\n"

              "\033[95m5.\033[0;0m Main Menu\n")

        selected_option = input("Select a number: ")

        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1,
max_value=5)

        if not status:

            print(value)

            return self.guest_menu(Hotel)
```

```
    actions = {
        1: self.create_new_booking,
        2: self.display_all_guests,
        3: self.lookup_guest_by_room,
        4: self.update_guest_by_room,
        5: Hotel.main_menu
    }

    action = actions.get(value)
    if action:
        action() if value == 5 else (action(), self.guest_menu(Hotel))

def display_all_guests(self):
    """Displays all guests and their associated room info in tabular format."""
    self.cursor.execute('''
        SELECT B.ROOM_ID, G.NAME, G.EMAIL, G.PHONE, G.LOYALTY_STATUS
        FROM GUESTS G
        JOIN BOOKINGS B ON G.GUEST_ID = B.GUEST_ID
    ''')
    guests = self.cursor.fetchall()
    headers = ["Room Number", "Name", "Email", "Phone", "Loyalty Status"]
    print(tabulate(guests, headers=headers, tablefmt="grid"))
```

```
    def insert_guest(self, name: str, email: str, phone: str, loyalty_status: str):
        """Inserts a new guest into the database."""
        self.cursor.execute('INSERT INTO GUESTS (NAME, EMAIL, PHONE, LOYALTY_STATUS) VALUES (?, ?, ?, ?)',
                            (name, email, phone, loyalty_status))
        self.conn.commit()


    def lookup_guest_by_room(self):
        """Finds and displays guest details and booking history based on room number."""
        room_number = input("Enter room number to look up guest: ")
        try:
            room_id = int(room_number)
            self.cursor.execute('''
                SELECT B.ROOM_ID, G.NAME, G.EMAIL, G.PHONE, G.LOYALTY_STATUS, G.GUEST_ID
                FROM GUESTS G
                JOIN BOOKINGS B ON G.GUEST_ID = B.GUEST_ID
                WHERE B.ROOM_ID = ?
            ''', (room_id,))
            guest = self.cursor.fetchone()
            if guest:
                guest_display = guest[:-1]  # Exclude GUEST_ID
                guest_id = guest[-1]
                headers = ["Room Number", "Name", "Email", "Phone", "Loyalty Status"]
```

```
            print(tabulate([guest_display], headers=headers, tablefmt="grid"))


            self.cursor.execute('''
                SELECT ROOM_ID, CHECK_IN_DATE, CHECK_OUT_DATE
                FROM BOOKINGS
                WHERE GUEST_ID = ?
                ORDER BY CHECK_IN_DATE DESC
            ''', (guest_id,))
            history = self.cursor.fetchall()
            if history:
                print("\nBooking History:")
                print(tabulate(history, headers=["Room ID", "Check-in Date", "Check-out Date"],
tablefmt="grid"))
            else:
                print("\nNo booking history found.")
        else:
            print("No guest found for that room.")
    except ValueError:
        print("Invalid room number entered.")


def update_guest_by_room(self):
    """Updates guest information for a given room number."""
    room_number = input("Enter room number to update guest details: ")
    try:
```

```
        room_id = int(room_number)
        self.cursor.execute('''
            SELECT G.GUEST_ID, G.NAME, G.EMAIL, G.PHONE, G.LOYALTY_STATUS
            FROM GUESTS G
            JOIN BOOKINGS B ON G.GUEST_ID = B.GUEST_ID
            WHERE B.ROOM_ID = ?
        ''', (room_id,))
        result = self.cursor.fetchone()
        if result:
            guest_id, name, email, phone, loyalty_status = result
            print("Leave input blank and press Enter to keep current value.")
            new_name = input(f"Name [\033[95m{name}\033[0;0m]: ") or name
            new_email = input(f"Email [\033[95m{email}\033[0;0m]: ") or email
            new_phone = input(f"Phone [\033[95m{phone}\033[0;0m]: ") or phone
            new_loyalty = input(f"Loyalty Status [\033[95m{loyalty_status}\033[0;0m]: ") or
loyalty_status

            self.cursor.execute('''
                UPDATE GUESTS
                SET NAME = ?, EMAIL = ?, PHONE = ?, LOYALTY_STATUS = ?
                WHERE GUEST_ID = ?
            ''', (new_name, new_email, new_phone, new_loyalty, guest_id))
            self.conn.commit()
            print("Guest information updated successfully.")
```

```
            else:
                print("No guest found for that room.")
        except ValueError:
            print("Invalid room number entered.")

    def create_new_booking(self):
        """Creates a new booking for a guest, inserting new guest if not found."""
        print("\n--- New Booking Form ---")
        name = input("Guest Name: ").strip()
        email = input("Email: ").strip().lower()
        phone = input("Phone (e.g., 05XXXXXXXX): ").strip()

        self.cursor.execute("SELECT GUEST_ID, NAME, EMAIL, PHONE,
LOYALTY_STATUS FROM GUESTS WHERE EMAIL = ?", (email,))
        guest = self.cursor.fetchone()

        if guest:
            guest_id, g_name, g_email, g_phone, g_loyalty = guest
            print("\nGuest found:")
            print(tabulate([guest], headers=["Guest ID", "Name", "Email", "Phone", "Loyalty
Status"], tablefmt="grid"))
            print("\nThank you for booking again!")
            if g_loyalty.lower() == "bronze":
                print("Eligible for Silver loyalty upgrade!")
```

```
            elif g_loyalty.lower() == "silver":
                print("You're close to Gold status!")
            elif g_loyalty.lower() == "gold":
                print("Welcome back, Gold member!")
        else:
            loyalty_status = "Bronze"
            self.insert_guest(name, email, phone, loyalty_status)
            self.cursor.execute("SELECT GUEST_ID FROM GUESTS WHERE EMAIL = ?", (email,))
            guest_id = self.cursor.fetchone()[0]
            print("\nNew guest added with Bronze loyalty status.")

        # Show and select available room
        available_rooms = self.query_rooms_by_status("Available")
        print("\nAvailable Rooms:")
        print(tabulate(available_rooms, headers=["Room ID", "Type", "Amenities", "Price", "Status"], tablefmt="grid"))

        try:
            room_id = int(input("Enter Room ID to book: "))
            check_in = input("Check-in date (YYYY-MM-DD): ")
            check_out = input("Check-out date (YYYY-MM-DD): ")
            self.book_room(guest_id, room_id, check_in, check_out)
            print("\nBooking created successfully!")
```

```
        except ValueError:
            print("Invalid room ID or date format.")

    def book_room(self, guest_id: int, room_id: int, check_in: str, check_out: str):
        """Books a room for a guest and updates the room status."""
        self.cursor.execute('''
            INSERT INTO BOOKINGS (GUEST_ID, ROOM_ID, CHECK_IN_DATE, CHECK_OUT_DATE)
            VALUES (?, ?, ?, ?)
        ''', (guest_id, room_id, check_in, check_out))
        self.cursor.execute('UPDATE ROOMS SET STATUS = ? WHERE ID = ?', ('Occupied', room_id))
        self.conn.commit()

    def query_rooms_by_status(self, status: str) -> List[Tuple[int, str, str, float, str]]:
        """Fetches rooms from the database by status."""
        self.cursor.execute('SELECT * FROM ROOMS WHERE STATUS = ?', (status,))
        return self.cursor.fetchall()

from typing import List, Tuple
from inputValidation import Validation
from tabulate import tabulate
```

```
class ServiceRequest:
    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn


    def guest_services_menu(self, Hotel):
        """Displays the guest services menu and handles user input."""
        print("\n\033[0;0mAvailable options:\n"
              "\033[95m1.\033[0;0m House Keeping\n"
              "\033[95m2.\033[0;0m Room Service\n"
              "\033[95m3.\033[0;0m Custom Request\n"
              "\033[95m4.\033[0;0m Main Menu\n")
        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1,
max_value=4)
        if not status:
            print(value)
            return self.guest_services_menu(Hotel)


        actions = {
            1: self.submit_house_keeping_service_request,
            2: self.submit_guest_room_service_request,
            3: self.submit_guest_custom_request,
            4: Hotel.main_menu
```

```
    }

    action = actions.get(value)
    if action:
        action() if value == 4 else (action(), self.guest_services_menu(Hotel))

def get_guest_id_by_room(self, room_id: int) -> int:
    """Fetches the latest guest ID for a given room, or returns None."""
    self.cursor.execute('''
        SELECT G.GUEST_ID FROM BOOKINGS B
        JOIN GUESTS G ON B.GUEST_ID = G.GUEST_ID
        WHERE B.ROOM_ID = ?
        ORDER BY B.CHECK_IN_DATE DESC
        LIMIT 1
    ''', (room_id,))
    result = self.cursor.fetchone()
    return result[0] if result else None

def submit_guest_custom_request(self):
    """Handles submission of a custom guest service request."""
    print("\n--- Submit Service Request ---")
    room_number = input("Enter room number: ").strip()
    try:
```

```
            room_id = int(room_number)
            guest_id = self.get_guest_id_by_room(room_id)
            if not guest_id:
                print("No guest currently associated with that room.")
                return

            request_type = input("Enter request type: ").strip()
            fee_input = input("Enter fee for the request (AED): ").strip()
            try:
                fee = float(fee_input)
            except ValueError:
                fee = 0.0
                print("Invalid fee input, defaulting to 0.0")

            self.cursor.execute('''
                INSERT INTO REQUESTS (GUEST_ID, ROOM_ID, REQUEST_TYPE, FEE)
                VALUES (?, ?, ?, ?)
            ''', (guest_id, room_id, request_type, fee))
            self.conn.commit()
            print("Request submitted successfully.")

        except ValueError:
            print("Invalid room number entered.")
```

```
def submit_guest_room_service_request(self):
    """Submits a pre-defined 'Room Service' request."""
    print("\n--- Submit Room Service Request ---")
    self._submit_standard_request(service_type="Room Service")

def submit_house_keeping_service_request(self):
    """Submits a pre-defined 'House Keeping' request."""
    print("\n--- Submit House Keeping Request ---")
    self._submit_standard_request(service_type="House Keeping")

def _submit_standard_request(self, service_type: str):
    """Handles shared logic for submitting predefined service requests."""
    room_number = input("Enter room number: ").strip()
    try:
        room_id = int(room_number)
        guest_id = self.get_guest_id_by_room(room_id)
        if not guest_id:
            print("No guest currently associated with that room.")
            return

        fee_input = input(f"Enter fee for the {service_type} request (AED): ").strip()
        try:
```

```
            fee = float(fee_input)
        except ValueError:
            fee = 0.0
            print("Invalid fee input, defaulting to 0.0")

        self.cursor.execute('''
            INSERT INTO REQUESTS (GUEST_ID, ROOM_ID, REQUEST_TYPE, FEE)
            VALUES (?, ?, ?, ?)
        ''', (guest_id, room_id, service_type, fee))
        self.conn.commit()
        print("Request submitted successfully.")

    except ValueError:
        print("Invalid room number entered.")


from inputValidation import Validation


class Hotel:
    """Represents the hotel managing rooms, guests, and bookings."""

    def __init__(self, Room, Guest, GuestServices, Accounting, Feedback):
        self.Room = Room
        self.Guest = Guest
        self.GuestServices = GuestServices
        self.Accounting = Accounting
        self.Feedback = Feedback

    def main_menu(self):
        """Displays the main menu and routes the user to the selected module."""
        print("\n\033[0;0mAvailable options:\n"
```

```
            "\033[95m1.\033[0;0m Rooms\n"
            "\033[95m2.\033[0;0m Bookings\n"
            "\033[95m3.\033[0;0m Guest Services\n"
            "\033[95m4.\033[0;0m Accounting\n"
            "\033[95m5.\033[0;0m Feedbacks\033[0;0m\n")

        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1,
max_value=5)

        if not status:
            print(value)
            return self.main_menu()

        print(chr(27) + "[2J")  # Clear screen

        menu_actions = {
            1: self.Room.rooms_menu,
            2: self.Guest.guest_menu,
            3: self.GuestServices.guest_services_menu,
            4: self.Accounting.accounting_menu,
            5: self.Feedback.feedback_menu,
        }

        action = menu_actions.get(value)
        if action:
            action(Hotel=self)




class Validation:

    def validate_text_input(input_text, min_value, max_value):
        """
        Validates if the input text can be converted to an integer within a given range.

        Parameters:
        - text (str): The input string to validate.
        - min_value (int): The minimum acceptable integer value.
        - max_value (int): The maximum acceptable integer value.

        Returns:
        - (bool, str): Tuple containing:
            - True and a success message if valid.
```

```
        - False and an error message otherwise.
        """
        try:
            number = float(input_text)
        except ValueError:
            return False, "Input is not a number."

        if not number.is_integer():
            return False, "Number is not a whole number."

        number = int(number)
        if number < min_value or number > max_value:
            return False, f"Number is not within the range {min_value} to {max_value}."

        return True, number




from datetime import datetime
from inputValidation import Validation
from tabulate import tabulate


class Payment:
    def __init__(self, conn):
        self.cursor = conn.cursor()
        self.conn = conn

    def accounting_menu(self, Hotel):
        """Displays the accounting menu and handles user input."""
        print("\n\033[0;0mAvailable options:\n"
              "\033[95m1.\033[0;0m Generate Invoice\n"
              "\033[95m2.\033[0;0m Guest Charges\n"
              "\033[95m3.\033[0;0m Main Menu\n")
        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1,
max_value=3)
        if not status:
            print(value)
            return self.accounting_menu(Hotel)

        actions = {
            1: self.generate_final_invoice,
            2: self.display_itemized_invoice,
```

```
            3: Hotel.main_menu
        }

        action = actions.get(value)
        if action:
            action() if value == 3 else (action(), self.accounting_menu(Hotel))

    def generate_final_invoice(self):
        """Generates and displays the final invoice for a guest and records payment."""
        print("\n--- Generate Final Invoice ---")
        room_number = input("Enter room number: ").strip()
        try:
            room_id = int(room_number)
            self.cursor.execute('''
                SELECT B.BOOKING_ID, G.GUEST_ID, R.PRICE, B.CHECK_IN_DATE,
B.CHECK_OUT_DATE
                FROM BOOKINGS B
                JOIN ROOMS R ON B.ROOM_ID = R.ID
                JOIN GUESTS G ON B.GUEST_ID = G.GUEST_ID
                WHERE B.ROOM_ID = ?
                ORDER BY B.CHECK_IN_DATE DESC
                LIMIT 1
            ''', (room_id,))
            booking = self.cursor.fetchone()

            if not booking:
                print("No booking found for that room.")
                return

            booking_id, guest_id, nightly_rate, check_in, check_out = booking
            num_nights = (datetime.strptime(check_out, "%Y-%m-%d") -
datetime.strptime(check_in, "%Y-%m-%d")).days
            stay_total = nightly_rate * num_nights

            self.cursor.execute('''
                SELECT SUM(FEE) FROM REQUESTS
                WHERE ROOM_ID = ? AND GUEST_ID = ?
            ''', (room_id, guest_id))
            request_fees = self.cursor.fetchone()[0] or 0.0
            total_due = stay_total + request_fees

            invoice_data = [[
                f"AED {nightly_rate:.2f} x {num_nights} nights = AED {stay_total:.2f}",
                f"Additional Requests = AED {request_fees:.2f}",
                f"TOTAL DUE = AED {total_due:.2f}"
            ]]
```

```
        print("\nFinal Invoice:")
        print(tabulate(invoice_data, headers=["Room Charges", "Service Charges", "Total"],
tablefmt="grid"))

        confirm = input("Proceed with credit card payment? (yes/no): ").strip().lower()
        if confirm == "yes":
            self.cursor.execute('''
                INSERT INTO ACCOUNTING (BOOKING_ID, NIGHTLY_RATE,
NUM_NIGHTS, ADDITIONAL_CHARGES, DISCOUNTS, TOTAL_AMOUNT,
PAYMENT_METHOD)
                VALUES (?, ?, ?, ?, ?, ?, ?)
            ''', (booking_id, nightly_rate, num_nights, request_fees, 0.0, total_due, "Credit Card"))
            self.conn.commit()
            print("\nPayment successful. Invoice saved.")
        else:
            print("\nPayment cancelled.")

    except ValueError:
        print("Invalid room number entered.")

  def display_itemized_invoice(self):
    """Displays itemized charges for a guest's stay including services used."""
    print("\n--- Itemized Invoice ---")
    room_number = input("Enter room number: ").strip()
    try:
        room_id = int(room_number)
        self.cursor.execute('''
            SELECT B.BOOKING_ID, G.GUEST_ID, R.PRICE, B.CHECK_IN_DATE,
B.CHECK_OUT_DATE
            FROM BOOKINGS B
            JOIN ROOMS R ON B.ROOM_ID = R.ID
            JOIN GUESTS G ON B.GUEST_ID = G.GUEST_ID
            WHERE B.ROOM_ID = ?
            ORDER BY B.CHECK_IN_DATE DESC
            LIMIT 1
        ''', (room_id,))
        booking = self.cursor.fetchone()

        if not booking:
            print("No booking found for that room.")
            return

        booking_id, guest_id, nightly_rate, check_in, check_out = booking
        num_nights = (datetime.strptime(check_out, "%Y-%m-%d") -
datetime.strptime(check_in, "%Y-%m-%d")).days
        stay_total = nightly_rate * num_nights
```

```
            self.cursor.execute('''
                SELECT REQUEST_TYPE, FEE FROM REQUESTS
                WHERE ROOM_ID = ? AND GUEST_ID = ?
            ''', (room_id, guest_id))
            requests = self.cursor.fetchall()

            print("\nRoom Charges:")
            room_charges = [[f"Night {i + 1}", f"AED {nightly_rate:.2f}"] for i in
range(num_nights)]
            print(tabulate(room_charges, headers=["Description", "Amount"], tablefmt="grid"))

            print("\nService Charges:")
            if requests:
                print(tabulate(requests, headers=["Service", "Fee"], tablefmt="grid"))
            else:
                print("No service charges found.")

            request_fees = sum(fee for _, fee in requests) if requests else 0.0
            total_due = stay_total + request_fees

            print("\nTotal Due:")
            summary = [["Room Charges", f"AED {stay_total:.2f}"],
                       ["Service Charges", f"AED {request_fees:.2f}"],
                       ["Total Amount", f"AED {total_due:.2f}"]]
            print(tabulate(summary, tablefmt="grid"))

        except ValueError:
            print("Invalid room number entered.")




from typing import List, Tuple
from inputValidation import Validation
from tabulate import tabulate  # For clean table formatting


class Room:
    """Represents a hotel room with number, type, price, and bookings."""

    def __init__(self, conn):
        self.cursor = conn.cursor()

    def rooms_menu(self, Hotel):
```

```
        """Displays the room management menu and handles user selection."""
        print("\n\033[0;0mRooms Menu Options:\n"
              "\033[95m1.\033[0;0m Available Rooms\n"
              "\033[95m2.\033[0;0m Booked Rooms\n"
              "\033[95m3.\033[0;0m Maintenance Rooms\n"
              "\033[95m4.\033[0;0m Search Room\n"
              "\033[95m5.\033[0;0m Main Menu\n")

        selected_option = input("Select a number: ")
        status, value = Validation.validate_text_input(input_text=selected_option, min_value=1,
max_value=5)

        if not status:
            print(value)
            return self.rooms_menu(Hotel)

        print(chr(27) + "[2J")  # Clear screen

        menu_actions = {
            1: self.print_available_rooms,
            2: self.print_booked_rooms,
            3: self.print_maintenance_rooms,
            4: self.search_room,
            5: Hotel.main_menu
        }

        action = menu_actions.get(value)
        if action:
            if value in {1, 2, 3}:
                action()
                self.rooms_menu(Hotel)
            elif value == 4:
                self.search_room()
                self.rooms_menu(Hotel)
            else:
                action()

    def print_available_rooms(self):
        """Prints all available rooms."""
        print("Available Rooms:")
        rooms = self.query_rooms_by_status('Available')
        self.display_rooms(rooms)

    def print_booked_rooms(self):
        """Prints all occupied/booked rooms."""
        print("Booked Rooms:")
```

```
    rooms = self.query_rooms_by_status('Occupied')
    self.display_rooms(rooms)

def print_maintenance_rooms(self):
    """Prints all rooms under maintenance."""
    print("Maintenance Rooms:")
    rooms = self.query_rooms_by_status('Maintenance')
    self.display_rooms(rooms)

def search_room(self):
    """Prompts for a room ID and displays its details."""
    try:
        room_id = int(input("Please enter a room number: "))
        self.print_room(room_id)
    except ValueError:
        print("Invalid room number entered.")

def print_room(self, room_id):
    """Prints detailed info about a specific room."""
    print(f"Room {room_id}:")
    self.cursor.execute('SELECT * FROM ROOMS WHERE ID = ?', (room_id,))
    room = self.cursor.fetchone()
    if room:
        headers = ['Room ID', 'Type', 'Amenities', 'Price', 'Status']
        print(tabulate([room], headers=headers, tablefmt='grid'))
    else:
        print("Room not found.")

def display_rooms(self, rooms: List[Tuple[int, str, str, float, str]]):
    """Helper to print a list of rooms in tabular format."""
    if rooms:
        headers = ['Room ID', 'Type', 'Amenities', 'Price', 'Status']
        print(tabulate(rooms, headers=headers, tablefmt='grid'))
    else:
        print("No rooms to display.")

def query_rooms_by_status(self, status: str) -> List[Tuple[int, str, str, float, str]]:
    """Returns all rooms with a specific status."""
    self.cursor.execute('SELECT * FROM ROOMS WHERE STATUS = ?', (status,))
    return self.cursor.fetchall()
```

L. GitHub Link

https://github.com/Aboood2220/Royal-Stay-Hotel-Management-System

M. Summary

In this project, I developed a terminal-based hotel management system using Python, applying core principles of object-oriented programming (OOP). Through this experience, I gained a deeper understanding of how real-world entities—such as hotels, rooms, and guests—can be modeled using classes and objects.

The architecture consists of distinct modules such as Room, Guest, Payment, and Feedback, each encapsulated within its own class. For example, the Room class includes methods like print_available_rooms() and query_rooms_by_status() to manage and display room availability. The Guest class supports functions like create_new_booking() and update_guest_by_room() to handle dynamic guest data and booking logic. These modules are all coordinated through a central Hotel controller class, which presents a unified interface via the main_menu() method.

Planning the system using a UML class diagram was particularly beneficial—it allowed me to map out the relationships between components and plan interactions before diving into the code. This upfront design work helped reduce complexity during implementation.

While integrating the modules, one challenge I encountered was ensuring smooth interaction between classes. For instance, passing control between Hotel.main_menu() and module-specific menus required careful management of dependencies and parameters. This taught me the importance of clean interfaces and separation of concerns.

Finally, by building this system step-by-step, I learned how OOP facilitates scalability, code reuse, and maintainability. I now feel much more confident in my ability to design and implement structured, modular software systems using Python.