A

COLLEGE OF VOCATIONAL STUDIES

(University of delhi)

Hotel Management System

(Software engineering project report)

Group description

Avika singh	**22013570010**
Shristy	**22013570025**
Riddhi luthra	**22013570053**
Anuradha	**22013570036**

Under the supervision of : **Mrs. Parul Chachra**

Date – 26-11-2024

ACKNOWLEDGEMENT

On the successful completion of our project HOTEL MANAGEMENT SYSTEM we would like to express our sincere gratitude to everyone who helped us in the completion of this project.

We are sincerely thankful to our project guide Mrs. Parul Chachra, for her interest, guidance and suggestions throughout the course of the project. We feel honored and privileged to work under her. She shared her vast pool of knowledge with us that helped us steer through all the difficulties with ease. This project would not have been possible without her guidance.

CERTIFICATE

This is to certify that the project titled "HOTEL MANAGEMENT SYSTEM " is submitted by AVIKA SINGH , RIDDHI LUTHRA, SHRISTY and ANURADHA under the supervision of Mrs. Parul Chachra in the academic year 2024-2025.

This project is submitted for paper Software Engineering of BSc (Hons.) Computer Science, Semester V.

Supervisor

Mrs. Parul Chachra

(Teacher In-Charge)

(Department: Computer Science)

TABLE OF CONTENTS

1. Introduction

1.1 Problem Statement-

The ‘Hotel Management System (HMS)’ is designed to streamline hotel operations by automating room bookings, customer management, and room tracking processes. The system addresses inefficiencies in manual processes, ensuring faster, error-free handling of bookings and customer details. Key functionalities include:

- Simplified room booking based on real-time availability.
- Management of customer details such as contact information and stay preferences.
- Automated calculation of room charges, taxes, and total bills.
- Generation of detailed invoices for customers.

1.2 Objectives

- Provide a robust, user-friendly platform for hotel staff to manage daily operations efficiently.
- Enable customers to experience seamless room booking and billing.
- Ensure accurate data handling and minimize errors in bookings.
- Create a scalable and maintainable system for future feature integration.

1.3 Intended Audience and Reading Suggestions

This document is intended for:

Project Managers: To understand the overall design and architecture of the HMS Developers: To comprehend the technical details of the system, including database management, API usage, and system modules.

Testers: To gain insight into the test cases and system behavior to ensure quality and reliability.

End Users: To provide guidance on how to interact with the system, particularly those responsible for daily operations like customer management, room bookings , generating invoices. It is suggested that readers first familiarize themselves with the System Overview and then proceed to the more technical sections like System Features and External Interface Requirements for deeper insights into the HMS.

1.4 Process Model: Iterative Development

The ITERATIVE MODEL was selected for its flexibility and feedback-driven approach. Each iteration results in a functional version of the system, which undergoes evaluation and refinement.

Benefits include:

- EARLY FEEDBACK : Stakeholders can provide input at each iteration, ensuring alignment with requirements.
- REDUCED RISKS : Issues are identified and resolved early in the development cycle.
- CONTINUOUS IMPROVEMENT: The system evolves incrementally, with functionality and usability enhancements in each cycle.

2. Overall Description

2.1Product Perspective

- The HMS replaces traditional manual systems with a modern, automated solution. It is a standalone system integrated with a central database to manage customer data, room information, and transaction records. The system uses a web-based or desktop interface for users to interact with the functionalities.

The system consists of the following key components:

- Customer Management: Handles customer details and booking preferences.
- Room Availability Check: Ensures rooms are booked only if available.
- Booking and Payment Processing: Calculates charges and generates invoices.
- Admin Operations: Allows staff to update room statuses and view reports.

2.2 Product Functions

The HMS provides the following functionalities:

Customer Management:

- Collect and validate customer details.
- Store customer preferences and booking history.

Room Booking:

- Allow customers to book rooms based on real-time availability.
- Confirm bookings and generate invoices.

Admin Operations:

- Update room availability (e.g., maintenance or booking status).
- Generate and view reports on booking trends and revenues.

Billing:

- Calculate total charges, including taxes.
- Generate and store invoices for customer records.

2.3 User Characteristics

The system is designed for the following types of users:

Customers:

- Individuals booking rooms for their stay(via admin) .
- Minimal technical knowledge required to interact with the system.

Admins:

- Hotel staff responsible for managing bookings and room statuses.
- Requires basic knowledge of hotel operations and system usage.

2.4 General Constraints

- The system must operate within the constraints of the hotel's existing IT infrastructure.

- Data processing must comply with data protection regulations such as GDPR.
- The interface should be optimized for desktop systems and compatible with modern browsers.
- The system should support up to 500 simultaneous users without performance degradation.

2.5 Assumptions and Dependencies

- The system assumes accurate initial data input regarding room availability and pricing.
- The hotel will provide a reliable network infrastructure to support the system.
- Integration with third-party payment gateways may be required in future versions.

2.6 Operating Environment

- Software: The HMS will run on a modern web browser (e.g., Chrome, Firefox) with a responsive design for desktop and mobile compatibility.
- Server: The system can run on any standard web server capable of supporting Python web frameworks such as Flask or Django.
- Database: The system uses MySQL for managing data related to customers, rooms, employees and invoice.
- Platform: The system is accessible through web-based interfaces on Windows,macOS , and Linux platforms

3.Data Framework

3.1 Data Dictionary –

The data dictionary is a centralized repository of information about data. It provides a detailed description of the data, including its meaning, relationship to other data, usage, and format.

Sno	Term	Description
1	Customer details	Includes name, gender, phone number, and email , address.
2	Booking Request	Room type, check-in/check-out dates.
3	Room Data Store	Stores room details, availability, pricing
4	Invoice details	Includes room charges, taxes and total bill amount.
5	Generated invoice	Save the booking invoice for future generation.

3.2 Definitions, Acronyms, and Abbreviations

Sno	Term	Description
1	HMS	Hotel Management System
2	Admin	Hotel staff responsible for managing bookings and rooms.
3	Invoice	A detailed bill containing room charges, taxes, and fees
4	Booking request	A customer's request to reserve a room
5	Availability check	A process to verify if a room is available for booking

4. Requirement Analysis -

4.1 Functional Requirements

Functional requirements are the desired operations of a software program or system, and they define how the system must respond to inputs. They are a key part of requirements analysis, which is the engineering field that deals with the design and maintenance of complex systems.

Customer Management:

- The system shall allow customers to enter personal details and booking requests.
- The system shall validate customer input for accuracy.

Room Availability Check:

- The system shall query the database for room availability based on user input.
- The system shall display feedback on room availability status.

Booking and Payment Processing:

- The system shall calculate charges based on room type, stay duration, and taxes.
- The system shall generate a detailed invoice upon booking confirmation.
- The system shall update the room status in the database after successful booking.

Admin Operations:

- The system shall allow admins to update room statuses (e.g., available, booked, under maintenance).
- The system shall provide reports on booking trends and revenue generation.

4.2 Non-Functional Requirements

Non-functional requirements (NFRs) are specifications that describe how a system should operate, focusing on quality attributes like performance, security, reliability, and scalability.

Performance:

- The system shall respond to user actions within 2 seconds under normal conditions.
- The system shall support up to 500 concurrent users.

Security:

- The system shall encrypt sensitive customer data, including personal and payment information.
- Admin operations shall require secure login credentials.
- Only authorized admins should be able to perform actions like deleting records or adding new rooms.

Scalability:

- The system shall be scalable to accommodate new features such as online payments and loyalty programs.

Usability:

- The interface shall be intuitive, with clear navigation and error handling.
- All text shall be displayed in English for international use.

Portability:

- The system should be compatible with the most commonly used browsers (Chrome, Firefox, Edge) and responsive on both desktop and mobile Devices.

4.3 External Interface Requirements

An “external interface requirement” refers to the specifications defining how a software system will interact with elements outside of itself, such as other systems, hardware devices, or users, including details like data formats, communication protocols, and user interface elements that are visible to the end-user.

User Interface:

- A web-based GUI for customers and admins to interact with the system.

Hardware Interface:

- Compatible with standard desktop computers and printers for invoice generation.

Software Interface:

- Integrated with a central database for storing and retrieving data.

5.Risk Management

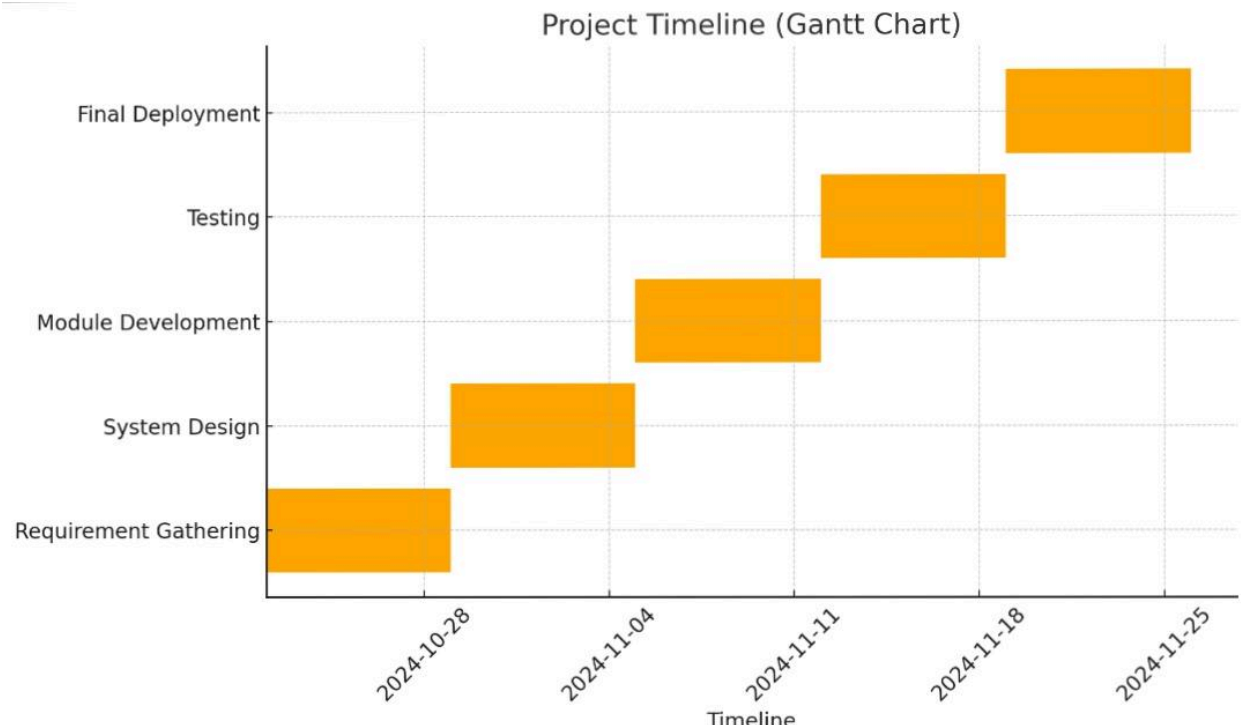
Risk	Likelihood	Impact	Mitigation Strategy

6.Project Management

6.1 Timeline (Gantt Chart)-

A popular type of timeline chart that shows a project's schedule using a horizontal bar chart. Each task is represented by a bar that runs from the start to the end date of the task. Gantt charts are useful for managing complex projects with dependencies.

Phase	Activities
Requirement Gathering	Identify stakeholders, gather system requirements, create SRS.
System Design	Design DFDs, ER diagrams, and finalize architecture.
Development	Implement features: Room booking, billing, admin panel.
Testing	Conduct functional, integration, and user acceptance testing.
Deployment and Handover	Deploy system, train staff, and finalize documentation.



6.2 Function Point (FP) Calculation-

FP calculation uses the Function Point Analysis (FPA) method based on complexity weightings for different system components.

Step 1: Assign Components –

Component	Type	Count	Weight (Simple, Avg, Complex)
External Inputs (EI)	Customer booking details	5	Simple (3)
External Outputs (EO)	Invoice generation	3	Average (4)
External Inquiries (EQ)	Room availability checks	5	Simple (3)
Internal Logical Files (ILF)	Customer and room data	4	Average (10)
External Interface Files (EIF)	Existing hotel data	2	Simple (7)

Step 2: Compute Unadjusted Function Points (UFP)-

Using the formula:

UFP=∑(count×weight)
UFP=∑(count×weight)
UFP=(5×3)+(3×4)+(5×3)+(4×10)+(2×7)
UFP=15+12+15+40+14=96

Step 3: Adjusted Function Points (AFP)-

Apply the Complexity Adjustment Factor (CAF). Assume an average system complexity with a CAF of 1.2:

AFP=UFP×CAF
AFP=96×1.2=115.2
Final FP: 115 (rounded)

6.3 Effort Estimation-

Effort estimation is based on the relationship:

Effort (Person-Months)=FP×Productivity (Hours/FP)

Assumptions:

- Average productivity = 10 hours/FP.
- Team works 160 hours/month.

Effort=(FP×Hours/FP)/(Hours/Month)

Effort=115×10/160

=7.1875 Person-Months (approx. 7 months)

Example:

If we have a team of 3 developers, the actual time required is:

7.18/3=2.39 months (approximately 10 weeks).

6.4 Cost Estimation-

Cost estimation uses the formula:

Cost=Effort (Person-Months)×Cost per Month

Assumptions:

- Average developer salary = $4,000/month.
- Cost=7.18×4,000=$28,720

Example:

For a team of 3 developers:

Cost=Developer Salary×Duration per Developer

Cost=(3×4,000)×2.39=$28,680

7 Design Engineering

7.1 Architectural Design

System Architecture: MVC (Model-View-Controller)

The 'Model-View-Controller (MVC)' pattern is used to organize the system into three interconnected layers for scalability and maintainability:

Model:

- Handles business logic, calculations, and database operations.
- Manages data related to rooms, bookings, and customers.

Example:

- Storing room details (availability, type, rate).
- Storing customers and booking information.

View:

- Represents the user interface (UI).
- Displays booking forms, availability status, and invoices.
- Ensures user-friendly interaction for both admins and customers.

Controller:

- Processes user inputs from the View and interacts with the Model.
- Ensures data flow between the UI and backend logic.

Example:

- Handling booking requests and forwarding them to the Model.
- Triggering invoice generation.

7.2 Component Diagram-

Component	Description
Login System	Handles user authentication (Admin).
Booking Management	Handles room availability checks and booking creation.
Billing Module	Calculates total cost (room charge, taxes, extras).
Room Management	Admin updates room details (availability, rates).
Database	Stores customer, booking, and room details securely.

7.3 Benefits of MVC and Modular Design

1. Scalability:

- Each layer (Model, View, Controller) can be updated independently.
- Easier integration of new features, like online payments.

2. Maintainability:

- Modular code reduces complexity.
- Clear separation of concerns ensures efficient debugging and updates.

3. Reusability:

- Components like `RoomBooking` can be reused for other hotel-related features.

8.Coding

8.1 Pseudocode for a Module-

Below is the pseudocode for a Room Booking Module, based on the previously provided structure and functions. The goal is to automate the booking process, validate availability, and calculate costs.

```
class RoomBooking:
    def __init__(self):
        self.rooms = {"Single": 5, "Double": 3, "Luxury": 2}  # Sample room data
        self.room_rates = {"Single": 100, "Double": 150, "Luxury": 250}  # Room rates
        self.tax_rate = 0.10  # 10% tax
```

```
    def check_availability(self, room_type):
        if room_type in self.rooms and self.rooms[room_type] > 0:
            return True
        return False

    def calculate_cost(self, room_type, num_days):
        if room_type in self.room_rates:
            subtotal = self.room_rates[room_type] * num_days
            tax = subtotal * self.tax_rate
            total_cost = subtotal + tax
            return {"subtotal": subtotal, "tax": tax, "total_cost": total_cost}
        return None

    def book_room(self, customer_name, room_type, num_days):
        if self.check_availability(room_type):
            self.rooms[room_type] -= 1  # Decrease available room count
            bill = self.calculate_cost(room_type, num_days)
            print(f"Booking Successful for {customer_name}!")
            print(f"Room Type: {room_type}")
            print(f"Number of Days: {num_days}")
            print(f"Subtotal: ${bill['subtotal']}")
            print(f"Tax: ${bill['tax']}")
            print(f"Total Cost: ${bill['total_cost']}")
        else:
            print(f"Sorry, {room_type} rooms are unavailable!")

# Example usage
booking_system = RoomBooking()
booking_system.book_room("Alice", "Luxury", 3)
```

8.2 Code of a Module in PYTHON

The Room Booking System is implemented in Python using the Tkinter library for the graphical user interface (GUI).This implementation aligns with the core functionality described, including room availability checks, booking management, and invoice generation.

```
class RoomBookingSystem:
    def __init__(self):
        self.rooms = {
            "Single": {"available": 10, "price": 2000},
            "Double": {"available": 5, "price": 3500},
            "Luxury": {"available": 3, "price": 8000},
        }
        self.tax_rate = 0.12
        self.bookings = []

    def check_availability(self, room_type, num_rooms):
        if room_type in self.rooms and self.rooms[room_type]["available"] >=
num_rooms:
            return True
        return False

    def calculate_cost(self, room_type, num_rooms, num_days):
        room_price = self.rooms[room_type]["price"]
        subtotal = room_price * num_rooms * num_days
        tax = subtotal * self.tax_rate
        total_cost = subtotal + tax
        return {"subtotal": subtotal, "tax": tax, "total_cost": total_cost}

    def book_room(self, customer_name, room_type, num_rooms, num_days):
        if not self.check_availability(room_type, num_rooms):
            return f"Sorry, {room_type} rooms are unavailable for your request."

        self.rooms[room_type]["available"] -= num_rooms
        cost_details = self.calculate_cost(room_type, num_rooms, num_days)
        booking = {
            "customer_name": customer_name,
            "room_type": room_type,
            "num_rooms": num_rooms,
            "num_days": num_days,
            "cost_details": cost_details,
        }
        self.bookings.append(booking)

        confirmation_message = (
            f"Booking Successful for {customer_name}!\n"
            f"Room Type: {room_type}\n"
            f"Number of Rooms: {num_rooms}\n"
            f"Number of Days: {num_days}\n"
            f"Subtotal: ₹{cost_details['subtotal']:.2f}\n"
```

```
            f"Tax (12%): ₹{cost_details['tax']:.2f}\n"
            f"Total Cost: ₹{cost_details['total_cost']:.2f}\n"
        )
        return confirmation_message

    def get_booking_summary(self):
        if not self.bookings:
            return "No bookings have been made yet."
        summary = "Booking Summary:\n"
        for idx, booking in enumerate(self.bookings, 1):
            summary += (
                f"{idx}. {booking['customer_name']} - {booking['room_type']} - "
                f"{booking['num_rooms']} room(s) for {booking['num_days']} days.\n"
            )
        return summary


if __name__ == "__main__":
    booking_system = RoomBookingSystem()
    print(booking_system.book_room("Avika", "Luxury", 1, 3))
    print(booking_system.book_room("Riddhi", "Single", 2, 5))
    print(booking_system.book_room("Anu", "Double", 6, 2))
    print("\n" + booking_system.get_booking_summary())
```

Explanation of the Code

1. Class Definition:

- `RoomBookingSystem` encapsulates all functionalities related to room booking, including inventory management, cost calculation, and booking storage.

2. Key Methods:

- `check_availability`: Checks if the requested room type and number are available.

- `calculate_cost`: Computes the total cost, including room charges and taxes.

- `book_room`: Handles booking, updates inventory, calculates costs, and generates confirmation messages.

- `get_booking_summary`: Retrieves a summary of all bookings for review.

3. Room Inventory:

- Maintains room availability and prices using a dictionary structure.

- Updates room availability dynamically upon successful bookings.

4. Example Usage:

- Demonstrates successful and failed bookings based on availability.

- Outputs a detailed confirmation message for successful bookings.

5. Scalability:

- The code is modular and can be extended to include additional features, such as:

  - Online payments.

  - Customer authentication.

  - Integration with a database.

Sample Output

Case 1: Successful Booking

```
Booking Successful for
Avika!
Room Type: Luxury
Number of Rooms: 1
Number of Days: 3
Subtotal: ₹24000.00
Tax (12%): ₹2880.00
Total Cost: ₹26880.00
```

```
Booking Successful for Riddhi!
Room Type: Single
Number of Rooms: 2
Number of Days: 5
Subtotal: ₹20000.00
Tax (12%): ₹2400.00
Total Cost: ₹22400.00
```

Case 2: Room Unavailable

```
Sorry, Double rooms are unavailable for your request.
```

Booking Summary

```
Booking Summary:
1. Avika - Luxury - 1 room(s) for 3 days.
2. Riddhi - Single - 2 room(s) for 5 days.
```

Code for Login:-

```
from tkinter import *
```

```
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
import mysql.connector
from hotel import HotelManagementSystem  # Importing the HotelManagementSystem class


def main():
    win = Tk()
    app = LoginWindow(win)
    win.mainloop()


class LoginWindow:
    def __init__(self, root):
        self.root = root
        self.root.title("Hotel Management System - Login")
        self.root.geometry("1550x800+0+0")


        # Background Image
        self.bg = ImageTk.PhotoImage(file=r"c:\Users\avika\OneDrive\Desktop\login page background.jpg")
        lbl_bg = Label(self.root, image=self.bg)
        lbl_bg.place(x=0, y=0, relwidth=1, relheight=1)


        # Login Frame
        frame = Frame(self.root, bg="black")
        frame.place(x=610, y=170, width=340, height=450)
```

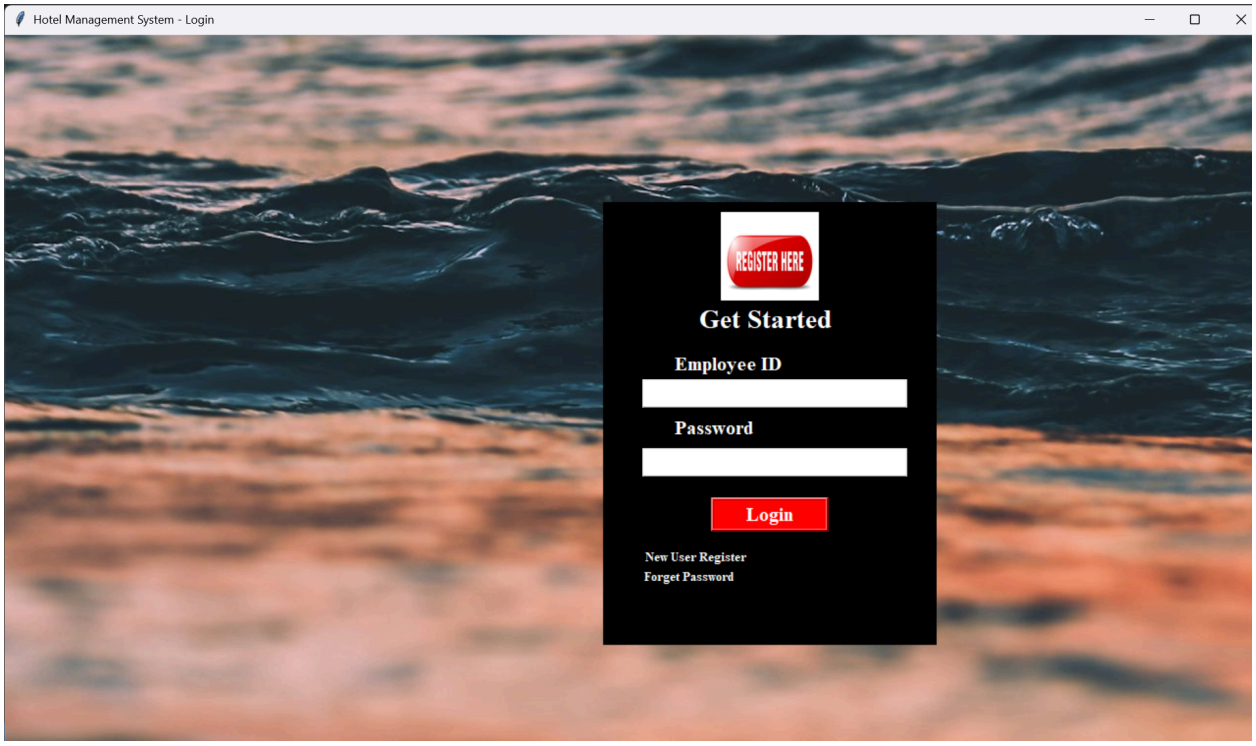
```
        # Login Logo
        img1 = Image.open(r"c:\Users\avika\OneDrive\Desktop\login.jpg")
        img1 = img1.resize((100, 100), Image.LANCZOS)
        self.photoimage1 = ImageTk.PhotoImage(img1)
        lblimg1 = Label(frame, image=self.photoimage1, bg="black",
borderwidth=0)
        lblimg1.place(x=120, y=10, width=100, height=100)


        # "Get Started" Label
        get_str = Label(frame, text="Get Started", font=("times new
roman", 20, "bold"), fg="white", bg="black")
        get_str.place(x=95, y=100)


        # Employee ID
        firstnamelbl = Label(frame, text="Employee ID", font=("times new
roman", 15, "bold"), fg="white", bg="black")
        firstnamelbl.place(x=70, y=150)
        self.txtuser = ttk.Entry(frame, font=("times new roman", 15,
"bold"))
        self.txtuser.place(x=40, y=180, width=270)


        # Password
        passwordlbl = Label(frame, text="Password", font=("times new
roman", 15, "bold"), fg="white", bg="black")
        passwordlbl.place(x=70, y=215)
        self.txtpass = ttk.Entry(frame, font=("times new roman", 15,
"bold"), show="*")
        self.txtpass.place(x=40, y=250, width=270)
```

```
        # Login Button
        loginbtn = Button(frame, command=self.login, text="Login",
font=("times new roman", 15, "bold"), bd=3,
                          relief=RIDGE, fg="white", bg="red",
activebackground="red")
        loginbtn.place(x=110, y=300, width=120, height=35)


        # Register Button
        registerbtn = Button(frame, text="New User Register", font=("times
new roman", 10, "bold"), bd=3, borderwidth=0,
                             fg="white", bg="black",
command=self.register_window)
        registerbtn.place(x=15, y=350, width=160)


        # Forgot Password Button
        pswrdbtn = Button(frame, text="Forget Password", font=("times new
roman", 10, "bold"), bd=3, borderwidth=0,
                          fg="white", bg="black")
        pswrdbtn.place(x=8, y=370, width=160)


    def login(self):
        """Verify login credentials."""
        if self.txtuser.get() == "" or self.txtpass.get() == "":
            messagebox.showerror("Error", "Please enter Employee ID and
Password")
        else:
            try:
                conn = mysql.connector.connect(
```

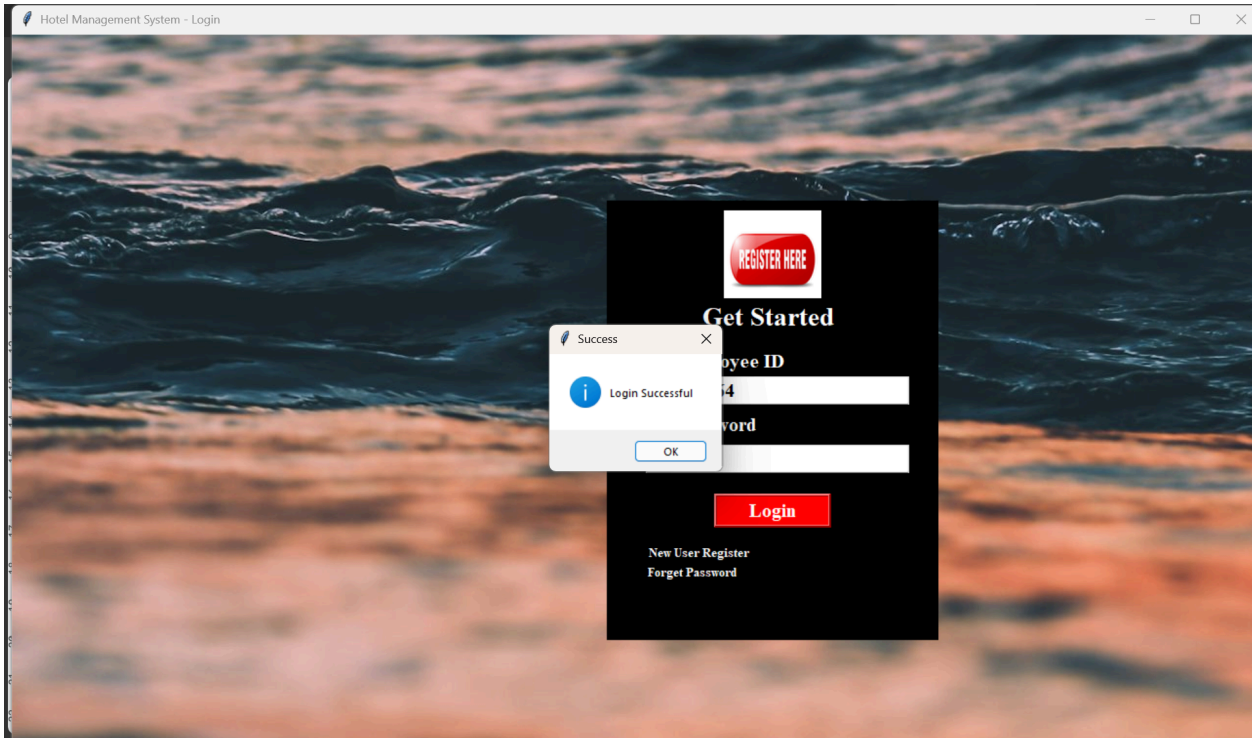
```
                    host="localhost", username="root", password="lambo",
database="HOTEL"
                )
                my_cursor = conn.cursor()
                emp_ref = self.txtuser.get().strip()
                password = self.txtpass.get().strip()

                # Query to verify credentials
                my_cursor.execute(
                    "SELECT * FROM register WHERE emp_ref=%s AND
password=%s",
                    (emp_ref, password),
                )
                row = my_cursor.fetchone()

                if row:
                    messagebox.showinfo("Success", "Login Successful")
                    self.open_hotel_management()  # Open the hotel
management system
                else:
                    messagebox.showerror("Error", "Invalid Employee ID or
Password")
            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Database error: {err}")
            finally:
                if conn.is_connected():
                    conn.close()
```

```
    def open_hotel_management(self):
        """Open the Hotel Management System interface."""
        self.new_window = Toplevel(self.root)
        HotelManagementSystem(self.new_window)  # Launch the Hotel
Management System
        self.root.withdraw()  # Hide the login window

    def register_window(self):
        """Open the registration window."""
        self.new_window = Toplevel(self.root)
        Register(self.new_window)
        self.root.withdraw()  # Hide the login window

class Register:
    def __init__(self, root):
        self.root = root
        self.root.title("Register")
        self.root.geometry("1550x800+0+0")
        # Similar to previous register window implementation
        # Define registration fields and database insertion logic here.

if __name__ == "__main__":
    main()
```

Output for this code ➖



when the login is successful after comparing with the database



Then the main interface of hotel:-

```
from tkinter import *
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
import mysql.connector  # Ensure MySQL connector is properly imported
from customer import Cust_Win
from room import Roombooking
from details import DetailsRoom



class LoginWindow:
    def __init__(self, root):
        self.root = root
```

```
        self.root.title("Hotel Management System - Login")
        self.root.geometry("1550x800+0+0")

        # Background Image
        self.bg = ImageTk.PhotoImage(file=r"c:\Users\avika\OneDrive\Desktop\login page background.jpg")
        lbl_bg = Label(self.root, image=self.bg)
        lbl_bg.place(x=0, y=0, relwidth=1, relheight=1)

        # Login Frame
        frame = Frame(self.root, bg="black")
        frame.place(x=610, y=170, width=340, height=450)

        # Login Logo
        img1 = Image.open(r"c:\Users\avika\OneDrive\Desktop\login.jpg")
        img1 = img1.resize((100, 100), Image.LANCZOS)
        self.photoimage1 = ImageTk.PhotoImage(img1)
        lblimg1 = Label(frame, image=self.photoimage1, bg="black", borderwidth=0)
        lblimg1.place(x=120, y=10, width=100, height=100)

        # "Get Started" Label
        get_str = Label(frame, text="Get Started", font=("times new roman", 20, "bold"), fg="white", bg="black")
        get_str.place(x=95, y=100)

        # Employee ID
```

```
        firstnamelbl = Label(frame, text="Employee ID", font=("times new
roman", 15, "bold"), fg="white", bg="black")

        firstnamelbl.place(x=70, y=150)

        self.txtuser = ttk.Entry(frame, font=("times new roman", 15,
"bold"))

        self.txtuser.place(x=40, y=180, width=270)


        # Password

        passwordlbl = Label(frame, text="Password", font=("times new
roman", 15, "bold"), fg="white", bg="black")

        passwordlbl.place(x=70, y=215)

        self.txtpass = ttk.Entry(frame, font=("times new roman", 15,
"bold"), show="*")

        self.txtpass.place(x=40, y=250, width=270)


        # Login Button

        loginbtn = Button(

            frame,

            command=self.login,

            text="Login",

            font=("times new roman", 15, "bold"),

            bd=3,

            relief=RIDGE,

            fg="white",

            bg="red",

            activebackground="red",

        )

        loginbtn.place(x=110, y=300, width=120, height=35)
```

```
    def login(self):
        """Verify login credentials."""
        if self.txtuser.get() == "" or self.txtpass.get() == "":
            messagebox.showerror("Error", "Please enter Employee ID and
Password")
        else:
            try:
                conn = mysql.connector.connect(
                    host="localhost", username="root", password="lambo",
database="HOTEL"
                )
                my_cursor = conn.cursor()
                emp_ref = self.txtuser.get().strip()
                password = self.txtpass.get().strip()

                # Query to verify credentials
                my_cursor.execute(
                    "SELECT * FROM register WHERE emp_ref=%s AND
password=%s",
                    (emp_ref, password),
                )
                row = my_cursor.fetchone()

                if row:
                    messagebox.showinfo("Success", "Login Successful")
                    self.open_hotel_management()  # Open the hotel
management system
```

```
                else:
                    messagebox.showerror("Error", "Invalid Employee ID or
Password")
            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Database error: {err}")
            finally:
                if conn.is_connected():
                    conn.close()

    def open_hotel_management(self):
        """Open the Hotel Management System."""
        self.root.destroy()
        root = Tk()
        app = HotelManagementSystem(root)
        root.mainloop()



class HotelManagementSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Hotel Management System")
        self.root.geometry("1550x800+0+0")

        # Define target width and height for the header image and logo
        self.img_width = 950  # Width for the main hotel image
        self.img_height = 300  # Height for the main hotel image
        self.logo_width = 300  # Width for the logo
```

```
        self.logo_height = 300  # Height for the logo

        # Load and display the main hotel image
        self.load_image(r"c:\Users\avika\OneDrive\Desktop\taj.jpg", self.img_width, self.img_height, 0)

        # Load and display the logo image next to the main image
        self.load_image(r"c:\Users\avika\OneDrive\Desktop\taj logo.jpg", self.logo_width, self.logo_height, 0, offset=self.img_width)

        # Create and position the title label below the images
        self.create_title_label()

    def load_image(self, image_path, width, height, y_position, offset=0):
        try:
            print(f"Attempting to load image: {image_path}")
            img = Image.open(image_path)
            img = img.resize((width, height), Image.LANCZOS)
            photoimg = ImageTk.PhotoImage(img)

            # Create a Label widget for the image
            lblimg = Label(self.root, image=photoimg, bd=4, relief=RIDGE)
            lblimg.place(x=offset, y=y_position, width=width, height=height)

            # Keep a reference to the photo image to prevent garbage collection
            lblimg.image = photoimg
```

```
            print(f"Successfully loaded image: {image_path}")  # Log
success
        except FileNotFoundError:
            print(f"Error: Image file '{image_path}' not found.")
        except Exception as e:
            print(f"An error occurred: {e}")  # Log any other exceptions


    def create_title_label(self):
        title_text = "Hotel Management System"
        lbl_title = Label(self.root, text=title_text, font=("Times New
Roman", 40, "bold"), bg="black", fg="gold", bd=4, relief=RIDGE)

        title_width = 1550  # Width of the window
        lbl_width = 800     # Width of the label
        x_position = (title_width - lbl_width) // 2

        lbl_title.place(x=x_position, y=self.img_height, width=lbl_width,
height=57)

        main_frame = Frame(self.root, bd=10, relief=RIDGE)
        main_frame.place(x=0, y=350, width=1250, height=400)

        lbl_menu = Label(main_frame, text="MENU", font=("Times New Roman",
20, "bold"), bg="black", fg="gold", bd=4, relief=RIDGE)
        lbl_menu.place(x=0, y=0, width=230)

        btn_frame = Frame(main_frame, bd=10, relief=RIDGE)
```

```
        btn_frame.place(x=0, y=35, width=228, height=210)


        cust_btn = Button(btn_frame, text="CUSTOMER",
command=self.Cust_details, width=22, font=("Times New Roman", 16, "bold"),
bg="black", fg="gold", bd=0, cursor="hand1")
        cust_btn.grid(row=0, column=0, pady=1)


        room_btn = Button(btn_frame, text="ROOM",
command=self.roombooking, width=22, font=("Times New Roman", 16, "bold"),
bg="black", fg="gold", bd=0, cursor="hand1")
        room_btn.grid(row=1, column=0, pady=1)


        details_btn = Button(btn_frame, text="DETAILS", width=22,
command=self.DetailsRoom, font=("Times New Roman", 16, "bold"),
bg="black", fg="gold", bd=0, cursor="hand1")
        details_btn.grid(row=2, column=0, pady=1)


        report_btn = Button(btn_frame, text="FEEDBACK", width=22,
font=("Times New Roman", 16, "bold"), bg="black", fg="gold", bd=0,
cursor="hand1")
        report_btn.grid(row=3, column=0, pady=1)


        logout_btn = Button(btn_frame, text="LOGOUT", width=22,
command=self.logout, font=("Times New Roman", 16, "bold"), bg="black",
fg="gold", bd=0, cursor="hand1")
        logout_btn.grid(row=4, column=0, pady=1)




        img3 = Image.open(r"c:\Users\avika\OneDrive\Desktop\view1.jpg")
        img3 = img3.resize((1310, 375), Image.LANCZOS)
```

```
        self.photoimg3 = ImageTk.PhotoImage(img3)


        lbling = Label(main_frame, image=self.photoimg3, bd=4,
relief=RIDGE)
        lbling.place(x=225, y=0, width=1310, height=375)


        img4 = Image.open(r"c:\Users\avika\OneDrive\Desktop\tajhall.jpg")
        img4 = img4.resize((230, 210), Image.LANCZOS)
        self.photoimg4 = ImageTk.PhotoImage(img4)


        lbling = Label(main_frame, image=self.photoimg4, bd=4,
relief=RIDGE)
        lbling.place(x=0, y=250, width=230, height=110)


    def Cust_details(self):
        self.new_window = Toplevel(self.root)
        self.app = Cust_Win(self.new_window)


    def roombooking(self):
        self.new_window = Toplevel(self.root)
        self.app = Roombooking(self.new_window)


    def DetailsRoom(self):
        self.new_window = Toplevel(self.root)
        self.app = DetailsRoom(self.new_window)


    def logout(self):
```

```
        result = messagebox.askyesno("Logout", "Are you sure you want to
log out?")
        if result:
            self.root.destroy()  # Close the hotel management system
window
            root = Tk()
            app = LoginWindow(root)  # Reopen the login window
            root.mainloop()



if __name__ == "__main__":
    root = Tk()
    app = LoginWindow(root)
    root.mainloop()
```

the output:-

The Code for Customer:-

```
from tkinter import *

from PIL import Image, ImageTk

from tkinter import ttk

import random

import mysql.connector

from tkinter import messagebox


class Cust_Win:

    def __init__(self, root):

        self.root = root

        self.root.title("Hotel Management System")

        self.root.geometry("1310x700")
```

```
        # Variables for entries
        self.var_Ref = StringVar()
        x = random.randint(1000, 9999)
        self.var_Ref.set(str(x))

        self.var_cust_Name = StringVar()
        self.var_Gender = StringVar()
        self.var_Mobile_No = StringVar()
        self.var_Email = StringVar()
        self.var_Id_proof = StringVar()
        self.var_Id_Number = StringVar()
        self.var_Address = StringVar()

        # Title Label
        lbl_title = Label(self.root, text="ADD CUSTOMER DETAILS",
font=("Arial", 18, "bold"), bg="black", fg="gold", bd=4, relief="ridge")
        lbl_title.place(x=0, y=0, width=1295, height=50)

        # Load and display the logo image
        img2 = Image.open(r"c:\Users\avika\OneDrive\Desktop\taj logo.jpg")
        img2 = img2.resize((400, 195), Image.LANCZOS)
        self.photoimg2 = ImageTk.PhotoImage(img2)
        lbl_img = Label(self.root, image=self.photoimg2, bd=4,
relief=RIDGE)
        lbl_img.place(x=0, y=45, width=350, height=195)

        # Label frame for customer details
```

```
        self.labelframeleft = LabelFrame(self.root, bd=2, relief=RIDGE,
text="CUSTOMER DETAILS", padx=2, font=("Arial", 14, "bold"))
        self.labelframeleft.place(x=5, y=240, width=350, height=550)


        # Customer details fields with initial values and read-only state
for Customer Ref
        fields = [
            ("Customer Ref", self.var_Ref, 0, "readonly"),
            ("Customer Name", self.var_cust_Name, 1, "normal"),
            ("Mobile No", self.var_Mobile_No, 2, "normal"),
            ("Email", self.var_Email, 3, "normal"),
            ("ID Proof", self.var_Id_proof, 5, "normal"),
            ("ID Number", self.var_Id_Number, 6, "normal"),
            ("Gender", self.var_Gender, 7, "normal"),
            ("Address", self.var_Address, 8, "normal"),
        ]


        entries = {}
        for text, variable, row, state in fields:
            lbl = Label(self.labelframeleft, text=text, font=("Arial", 12,
"bold"), padx=5, pady=8)
            lbl.grid(row=row, column=0, sticky=W)
            if text in ["ID Proof", "Gender"]:
                combo = ttk.Combobox(self.labelframeleft,
textvariable=variable, font=("Arial", 12), state="readonly", width=13)
                if text == "ID Proof":
                    combo['values'] = ("Select", "Aadhar Card", "Driving
License", "Other")
                elif text == "Gender":
```

```
                    combo['values'] = ("Select", "Male", "Female",
"Other")
                combo.current(0)
                combo.grid(row=row, column=1, padx=10, pady=6)
                entries[text] = combo
            else:
                entry = ttk.Entry(self.labelframeleft,
textvariable=variable, width=15, font=("Arial", 12), state=state)
                entry.grid(row=row, column=1, padx=10, pady=6)
                entries[text] = entry

        # Buttons
        btn_frame = Frame(self.labelframeleft, bd=2, relief=RIDGE)
        btn_frame.place(x=0, y=400, width=380, height=40)
        buttons = [
            ("Save", "green", self.save_data),
            ("Clear", "blue", self.clear_data),
            ("Delete", "red", self.delete_data),
            ("Update", "black", self.update_data)  # Update button added
        ]
        for text, color, command in buttons:
            btn = Button(btn_frame, text=text, font=("Arial", 12, "bold"),
bg=color, fg="white", command=command)
            btn.pack(side=LEFT, padx=5, fill=X, expand=True)

        # View and Search frame
        Table_Frame = LabelFrame(self.root, bd=2, relief=RIDGE, text="VIEW
DETAILS AND SEARCH SYSTEM", font=("Arial", 19, "bold"), bg="black",
fg="white", padx=2, pady=6)
```

```
        Table_Frame.place(x=360, y=70, width=860, height=650)


        lbl_SearchBy = Label(Table_Frame, text="Search By:",
font=("Arial", 15, "bold"), bg="gold", fg="black")
        lbl_SearchBy.grid(row=0, column=0, sticky=W)
        self.search_var = StringVar()


        self.combo_Search = ttk.Combobox(Table_Frame, font=("Arial", 12),
state="readonly", width=24)
        self.combo_Search['values'] = ("Mobile", "Ref")
        self.combo_Search.current(0)
        self.combo_Search.grid(row=0, column=1, padx=10, pady=6)
        self.txt_search = StringVar()


        self.txtSearch = ttk.Entry(Table_Frame,
textvariable=self.txt_search, width=24, font=("Arial", 12, "bold"))
        self.txtSearch.grid(row=0, column=2, padx=3, pady=6)


        # Search and Show All buttons
        btn_search = Button(Table_Frame, text="Search",
command=self.search_data, font=("Arial", 15, "bold"), bg="gold",
fg="black")
        btn_search.grid(row=0, column=3, padx=2)


        btn_ShowAll = Button(Table_Frame, text="Show All",
command=self.fetch_data, font=("Arial", 15, "bold"), bg="gold",
fg="black")
        btn_ShowAll.grid(row=0, column=4, padx=2)
```

```
        # Table Frame for details
        details_table = Frame(Table_Frame, bd=2, relief=RIDGE)
        details_table.place(x=0, y=60, width=850, height=500)

        # Scrollbars
        scroll_x = ttk.Scrollbar(details_table, orient=HORIZONTAL)
        scroll_y = ttk.Scrollbar(details_table, orient=VERTICAL)
        scroll_x.pack(side=BOTTOM, fill=X)
        scroll_y.pack(side=RIGHT, fill=Y)

        # Treeview configuration
        self.Cust_details_Table = ttk.Treeview(
            details_table,
            columns=("Ref", "Name", "Gender", "Mobile No.", "Email",
"Idproof", "Id Number", "Address"),
            xscrollcommand=scroll_x.set,
            yscrollcommand=scroll_y.set
        )
        scroll_x.config(command=self.Cust_details_Table.xview)
        scroll_y.config(command=self.Cust_details_Table.yview)

        # Column configuration for Treeview
        for col in ("Ref", "Name", "Gender", "Mobile No.", "Email",
"Idproof", "Id Number", "Address"):
            self.Cust_details_Table.heading(col, text=col)
            self.Cust_details_Table.column(col, width=100)

        self.Cust_details_Table["show"] = "headings"
```

```
        self.Cust_details_Table.pack(fill=BOTH, expand=True)


        self.Cust_details_Table.bind("<ButtonRelease-1>", self.get_cursor)


        # Fetch initial data
        self.fetch_data()


    def save_data(self):
        if self.var_Mobile_No.get() == "" or self.var_Email.get() == "":
            messagebox.showerror("Error", "Mobile No. and Email are
required fields.", parent=self.root)
            return


        # Save new customer data
        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("INSERT INTO customer (ref, Name, Gender,
MobileNo, Email, Idproof, IdNumber, Address) VALUES (%s, %s, %s, %s, %s,
%s, %s, %s)", (
                self.var_Ref.get(),
                self.var_cust_Name.get(),
                self.var_Gender.get(),
                self.var_Mobile_No.get(),
                self.var_Email.get(),
                self.var_Id_proof.get(),
                self.var_Id_Number.get(),
```

```
                self.var_Address.get()

            ))

            conn.commit()

            messagebox.showinfo("Success", "Customer details have been
saved", parent=self.root)

            self.clear_data()

        except Exception as es:

            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)

        finally:

            conn.close()


    def update_data(self):

        # Update existing customer data

        try:

            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")

            my_cursor = conn.cursor()

            my_cursor.execute("UPDATE customer SET Name=%s, Gender=%s,
MobileNo=%s, Email=%s, Idproof=%s, IdNumber=%s, Address=%s WHERE ref=%s",
(

                self.var_cust_Name.get(),

                self.var_Gender.get(),

                self.var_Mobile_No.get(),

                self.var_Email.get(),

                self.var_Id_proof.get(),

                self.var_Id_Number.get(),

                self.var_Address.get(),

                self.var_Ref.get()
```

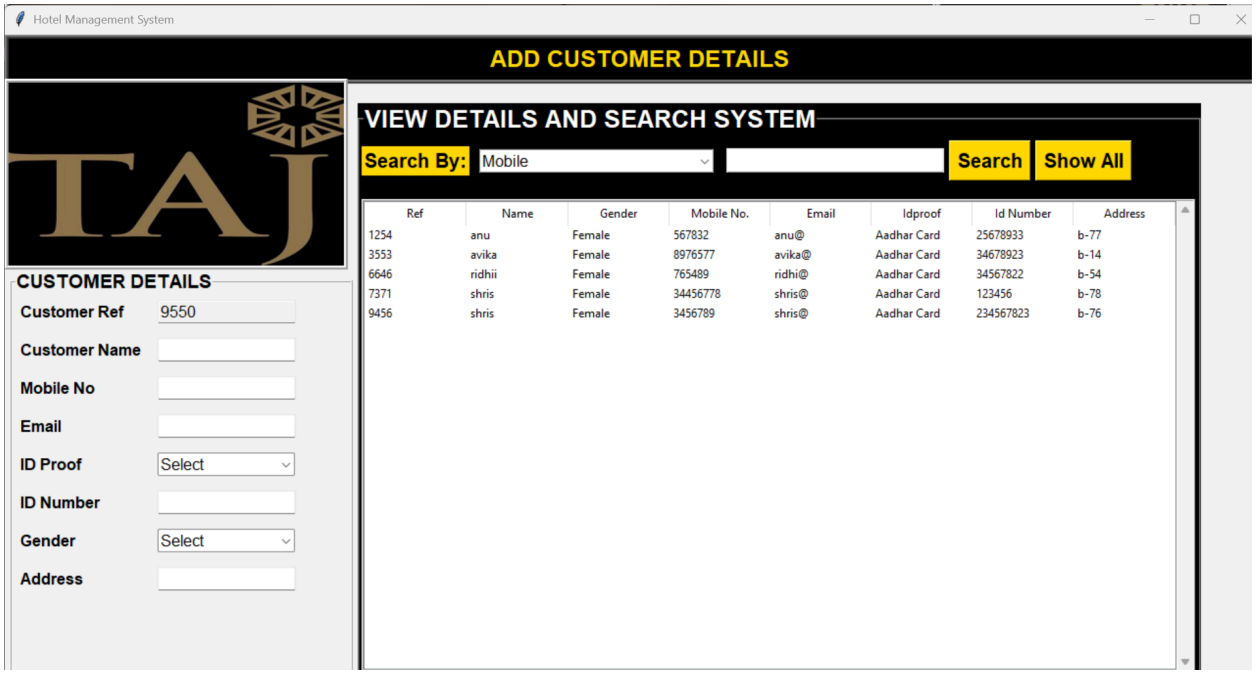
```
            ))

            conn.commit()

            messagebox.showinfo("Success", "Customer details have been
updated", parent=self.root)

            self.clear_data()

        except Exception as es:

            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)

        finally:

            conn.close()


    def clear_data(self):

        x = random.randint(1000, 9999)  # Generate a new random reference
number

        self.var_Ref.set(str(x))  # Set the new reference number

        self.var_cust_Name.set("")

        self.var_Gender.set("Select")

        self.var_Mobile_No.set("")

        self.var_Email.set("")

        self.var_Id_proof.set("Select")

        self.var_Id_Number.set("")

        self.var_Address.set("")

        self.fetch_data()  # Refresh the data view


    def delete_data(self):

        if self.var_Ref.get() == "":

            messagebox.showerror("Error", "Select a customer record to
delete", parent=self.root)
```

```
            return
        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("DELETE FROM customer WHERE ref=%s",
(self.var_Ref.get(),))
            conn.commit()
            messagebox.showinfo("Success", "Customer has been deleted",
parent=self.root)
            self.clear_data()
        except Exception as es:
            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)
        finally:
            conn.close()


    def fetch_data(self):
        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("SELECT * FROM customer")
            rows = my_cursor.fetchall()
            if len(rows) != 0:

self.Cust_details_Table.delete(*self.Cust_details_Table.get_children())
                for row in rows:
                    self.Cust_details_Table.insert("", END, values=row)
```

```
                conn.commit()
        except Exception as es:
            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)
        finally:
            conn.close()


    def search_data(self):
        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            query = "SELECT * FROM customer WHERE " +
self.combo_Search.get() + " LIKE %s"
            value = "%" + self.txt_search.get() + "%"
            my_cursor.execute(query, (value,))
            rows = my_cursor.fetchall()
            if len(rows) != 0:

self.Cust_details_Table.delete(*self.Cust_details_Table.get_children())
                for row in rows:
                    self.Cust_details_Table.insert("", END, values=row)
                conn.commit()
        except Exception as es:
            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)
        finally:
            conn.close()
```

```
    def get_cursor(self, event):
        cursor_row = self.Cust_details_Table.focus()
        content = self.Cust_details_Table.item(cursor_row)
        row = content['values']
        self.var_Ref.set(row[0])
        self.var_cust_Name.set(row[1])
        self.var_Gender.set(row[2])
        self.var_Mobile_No.set(row[3])
        self.var_Email.set(row[4])
        self.var_Id_proof.set(row[5])
        self.var_Id_Number.set(row[6])
        self.var_Address.set(row[7])


if __name__ == "__main__":
    root = Tk()
    obj = Cust_Win(root)
    root.mainloop()
```

Customer Interface:-



Code for booking the Details:-

```
from tkinter import *

from PIL import Image, ImageTk

from tkinter import ttk

import mysql.connector

from tkinter import messagebox


class DetailsRoom:

    def __init__(self, root):

        self.root = root

        self.root.title("Hotel Management System")

        self.root.geometry("1295x550+230+220")
```

```
        # Title Label
        lbl_title = Label(self.root, text="Room Booking Details",
font=("Arial", 18, "bold"), bg="black", fg="gold", bd=4, relief="ridge")
        lbl_title.place(x=5, y=0, width=1495, height=50)

        # Load and display the logo image
        img2 = Image.open(r"c:\Users\avika\OneDrive\Desktop\taj logo.jpg")
        img2 = img2.resize((100, 40), Image.LANCZOS)  # LANCZOS?
        self.photoimg2 = ImageTk.PhotoImage(img2)
        lbl_img = Label(self.root, image=self.photoimg2, bd=4,
relief=RIDGE)
        lbl_img.place(x=5, y=50, width=100, height=40)

        # Label frame for room details
        self.labelframeleft = LabelFrame(self.root, bd=2, relief=RIDGE,
text="New Room Add", font=("Arial", 12, "bold"), padx=2)
        self.labelframeleft.place(x=5, y=100, width=540, height=350)

        lbl_floor = Label(self.labelframeleft, text="Floor",
font=("arial", 12, "bold"), padx=2, pady=6)
        lbl_floor.grid(row=0, column=0, sticky=W)

        self.var_floor = StringVar()
        self.enty_floor = ttk.Entry(self.labelframeleft,
textvariable=self.var_floor, font=("arial", 13, "bold"), width=29)
        self.enty_floor.grid(row=0, column=1)
```

```
        lbl_roomno = Label(self.labelframeleft, text="Room No",
font=("arial", 12, "bold"), padx=2, pady=6)

        lbl_roomno.grid(row=1, column=0, sticky=W)



        self.var_roomno = StringVar()

        self.enty_Roomno = ttk.Entry(self.labelframeleft,
textvariable=self.var_roomno, font=("arial", 13, "bold"), width=29)

        self.enty_Roomno.grid(row=1, column=1)



        lbl_roomtype = Label(self.labelframeleft, text="Room Type",
font=("arial", 12, "bold"), padx=2, pady=6)

        lbl_roomtype.grid(row=2, column=0, sticky=W)



        self.var_roomtype = StringVar()

        self.room_type_combo = ttk.Combobox(self.labelframeleft,
textvariable=self.var_roomtype, font=("arial", 13, "bold"), width=27,
state="readonly")

        self.room_type_combo['values'] = ('Single', 'Double', 'Luxury')  #
Options for the combo box

        self.room_type_combo.grid(row=2, column=1)

        self.room_type_combo.set('Single')  # Default selection



        btn_frame = Frame(self.labelframeleft, bd=2, relief=RIDGE, pady=2)

        btn_frame.place(x=0, y=200, width=412, height=40)



        btnAdd = Button(btn_frame, text="Add", font=("arial", 11, "bold"),
bg="black", fg="gold", width=10, command=self.add_data)

        btnAdd.grid(row=0, column=0, padx=1)
```

```
        btnUpdate = Button(btn_frame, text="Update", font=("arial", 11,
"bold"), bg="black", fg="gold", width=10, command=self.update_data)
        btnUpdate.grid(row=0, column=1, padx=1)

        btnDelete = Button(btn_frame, text="Delete", font=("arial", 11,
"bold"), bg="black", fg="gold", width=10, command=self.delete_data)
        btnDelete.grid(row=0, column=2, padx=1)

        btnReset = Button(btn_frame, text="Reset", font=("arial", 11,
"bold"), bg="black", fg="gold", width=10, command=self.reset_data)
        btnReset.grid(row=0, column=3, padx=2, pady=2)

        # Table Frame
        Table_Frame = LabelFrame(self.root, bd=2, relief=RIDGE, text="Show
Room Details", font=("Arial", 19, "bold"), padx=2, pady=6)
        Table_Frame.place(x=600, y=55, width=600, height=350)

        scroll_x = ttk.Scrollbar(Table_Frame, orient=HORIZONTAL)
        scroll_y = ttk.Scrollbar(Table_Frame, orient=VERTICAL)
        scroll_x.pack(side=BOTTOM, fill=X)
        scroll_y.pack(side=RIGHT, fill=Y)

        # Treeview configuration
        self.room_table = ttk.Treeview(
            Table_Frame,
            columns=("Floor", "RoomNo", "RoomType"),
            xscrollcommand=scroll_x.set,
            yscrollcommand=scroll_y.set
```

```
        )
        scroll_x.config(command=self.room_table.xview)
        scroll_y.config(command=self.room_table.yview)

        # Column configuration for Treeview
        for col in ("Floor", "RoomNo", "RoomType"):
            self.room_table.heading(col, text=col)
            self.room_table.column(col, width=100)

        self.room_table["show"] = "headings"
        self.room_table.pack(fill=BOTH, expand=True)

        self.fetch_data()

    def add_data(self):
        if self.var_floor.get() == "" or self.var_roomno.get() == "" or
self.var_roomtype.get() == "":
            messagebox.showerror("Error", "All fields are required",
parent=self.root)
        else:
            try:
                conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
                my_cursor = conn.cursor()
                my_cursor.execute("INSERT INTO room_details (Floor,
RoomNo, RoomType) VALUES (%s, %s, %s)", (
                    self.var_floor.get(),
                    self.var_roomno.get(),
```

```
                    self.var_roomtype.get(),
                ))
                conn.commit()
                self.fetch_data()
                messagebox.showinfo("Success", "New room added",
parent=self.root)
                self.clear_data()
            except Exception as es:
                messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)
            finally:
                conn.close()


    def fetch_data(self):
        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("SELECT * FROM room_details")
            rows = my_cursor.fetchall()
            if len(rows) != 0:
                self.room_table.delete(*self.room_table.get_children())
                for row in rows:
                    self.room_table.insert("", END, values=row)
                conn.commit()
            conn.close()
        except Exception as es:
```

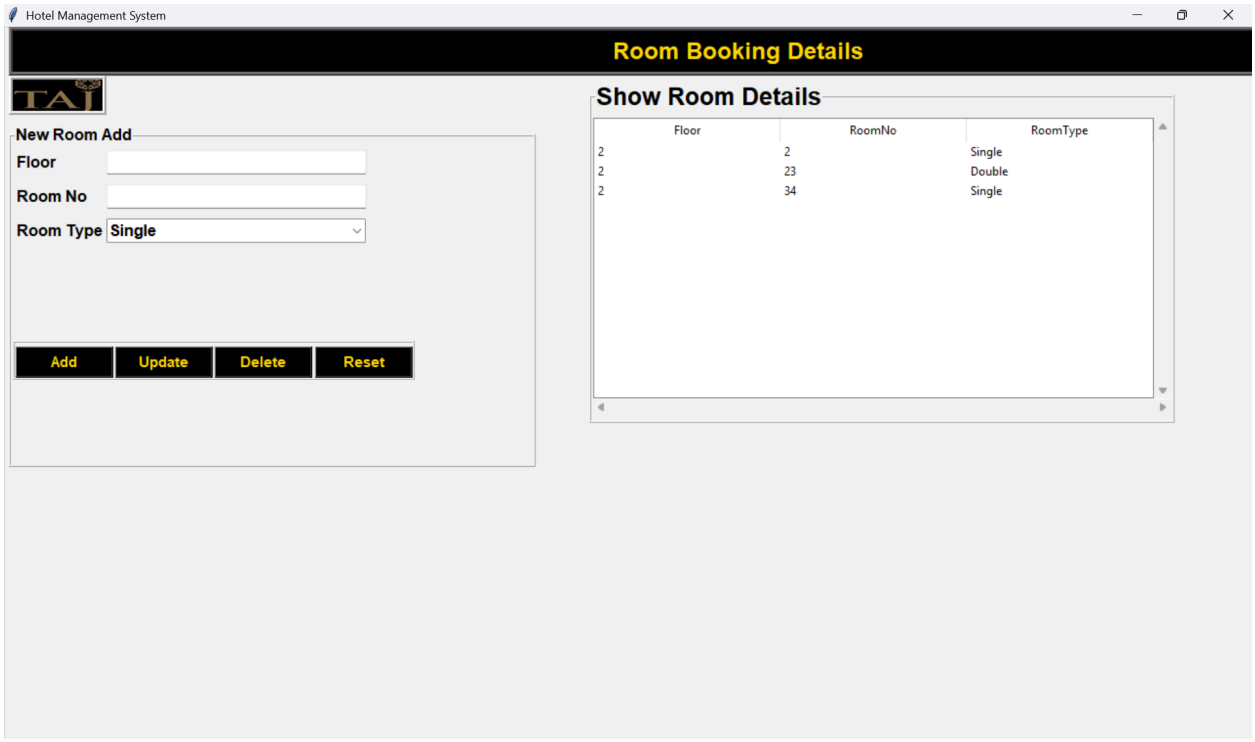
```
            messagebox.showwarning("Warning", f"Error fetching data:
{es}", parent=self.root)


    def update_data(self):
        print("Update button clicked")  # Debugging line
        selected_item = self.room_table.selection()
        if not selected_item:
            messagebox.showerror("Error", "Please select a room to
update", parent=self.root)
            return


        # Ensure the RoomNo is a valid integer
        room_no_str = self.var_roomno.get().strip()  # Strip any extra
spaces from the input


        # Check if RoomNo is a valid integer
        if not room_no_str.isdigit():
            messagebox.showerror("Error", "Room No must be a valid
integer", parent=self.root)
            return


        room_no = int(room_no_str)  # Convert the cleaned-up string to an
integer


        # Get the current room number from the selected row in the table
        current_room_no = self.room_table.item(selected_item)['values'][1]


        # If the user is changing the RoomNo, check if the new RoomNo
already exists
```

```
        if room_no != current_room_no:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("SELECT * FROM room_details WHERE
RoomNo=%s", (room_no,))
            existing_room = my_cursor.fetchone()
            if existing_room:
                messagebox.showerror("Error", "RoomNo already exists",
parent=self.root)
                conn.close()
                return

        # Now proceed to update the room details
        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("UPDATE room_details SET Floor=%s,
RoomNo=%s, RoomType=%s WHERE RoomNo=%s", (
                self.var_floor.get(),
                self.var_roomno.get(),
                self.var_roomtype.get(),
                current_room_no
            ))
            conn.commit()
            self.fetch_data()
            messagebox.showinfo("Success", "Room details updated",
parent=self.root)
```

```
            self.clear_data()
        except Exception as es:
            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)
        finally:
            conn.close()


    def delete_data(self):
        selected_item = self.room_table.selection()
        if not selected_item:
            messagebox.showerror("Error", "Please select a room to
delete", parent=self.root)
            return


        room_no = self.room_table.item(selected_item)['values'][1]


        try:
            conn = mysql.connector.connect(host="localhost",
username="root", password="lambo", database="HOTEL")
            my_cursor = conn.cursor()
            my_cursor.execute("DELETE FROM room_details WHERE RoomNo=%s",
(room_no,))
            conn.commit()
            self.fetch_data()
            messagebox.showinfo("Success", "Room deleted",
parent=self.root)
            self.clear_data()
        except Exception as es:
```

```
            messagebox.showwarning("Warning", f"Something went wrong:
{es}", parent=self.root)
        finally:
            conn.close()

    def reset_data(self):
        self.clear_data()

    def clear_data(self):
        self.var_floor.set("")
        self.var_roomno.set("")
        self.var_roomtype.set("Single")



# Create the root window and run the application
root = Tk()
obj = DetailsRoom(root)
root.mainloop()
```

Room Booking Interface ➖


Hotel Management System
Room Booking Details
TAJ
New Room Add
Floor
Room No
Room Type
Single
Add
Update
Delete
Reset
Show Room Details
Floor
RoomNo
RoomType
2
2
Single
2
23
Double
2
34
Single

9. Testing

9.1. Compute Basis Path for the Room Booking Module

Control Flow Graph (CFG) Representation of `book_room` Method:

1. Start.

2. Input customer details, room type, number of rooms, and number of days.

3. Check room availability (`check_availability` method).

   Decision 1 : If room is available, continue. Else, output “Room Unavailable” and terminate.

4. Calculate room cost (`calculate_cost` method).

5. Update room availability.

6. Generate invoice and confirmation message

7. End.

Independent Paths:

Using Cyclomatic Complexity (V(G)), the number of independent paths is calculated as:

V(G) = E – N + 2P

= Number of edges = 7,

= Number of nodes = 7,

= Number of connected components = 1.

V(G) = 7 – 7 + 2 = 2

Basis Paths:

1. Path 1: 1 → 2 → 3 → 4 → 5 → 6 → 7 (Successful booking).
2. Path 2: 1 → 2 → 3 → 7 (Room unavailable).

9.2. Generate Test Cases

9.2.1 Using white box

White box testing is a software testing method that involves examining a software system's internal structure, code, and logic. Testers have complete access to the application's design documents and source code, allowing them to understand how the software works and how it achieves its results.

Test Case 1: Successful Booking Path (Path 1)

Input:

Customer Name: “Avika”.

Room Type: “Luxury”.

Number of Rooms: 1.

Number of Days: 3.

Execution:

1. System checks availability (check_availability returns True).
2. Calculates total cost (₹8000/room * 3 days + 12% tax).
3. Updates inventory.
4. Generates booking confirmation.

Expected Output:

Booking Successful for Avika!

Room Type: Luxury

Number of Rooms: 1

Number of Days: 3

Subtotal: ₹24000.00

Tax (12%): ₹2880.00

Total Cost: ₹26880.00

Test Case 2: Room Unavailability Path (Path 2)

Input:

Customer Name: “Riddhi”.

Room Type: “Double”.

Number of Rooms: 6.

Number of Days: 2.

Execution:

1. System checks availability (check_availability returns False).

2. Booking process terminates with an unavailability message.

Expected Output:

Sorry, Double rooms are unavailable for your request.

Test Case 3: Invalid Input (Boundary Testing)

Input:

Customer Name: “Anu”.

Room Type: “Single”.

Number of Rooms: 0 (invalid input).

Number of Days: 3.

Execution:

1. Input validation identifies an invalid room count.
2. Process terminates with an error message.

Expected Output:

Error: Invalid input for number of rooms. Please enter a positive value.

Test Case 4: Extreme Values

Input:

Customer Name: “Riddhi”.

Room Type: “Luxury”.

Number of Rooms: 3 (maximum available).

Number of Days: 365 (long stay).

Execution:

1. System checks availability (check_availability returns True).
2. Calculates total cost for an extended stay.
3. Updates inventory.
4. Generates confirmation messages.

Expected Output:

Booking Successful for Riddhi!

Room Type: Luxury

Number of Rooms: 3

Number of Days: 365

Subtotal: ₹8760000.00

Tax (12%): ₹1051200.00

Total Cost: ₹9811200.00

9.2.2 Using black box

Black box testing is a software testing technique that assesses a system's functionality without knowing how it's coded or designed. Testers provide inputs and observe the system's outputs to evaluate how it responds to user actions, its performance, and other aspects.

Test Case 1: Valid Room Booking

Input:

Room Type: "Single".

Number of Rooms: 2.

Check-in Date: "2024-12-01".

Check-out Date: "2024-12-05".

Expected Behavior:

1. System verifies availability for two single rooms for 4 nights.
2. Calculates the total cost (₹2000/room * 4 nights + 12% tax).
3. Generates a confirmation message.

Expected Output:

Booking Successful!

Room Type: Single

Number of Rooms: 2

Total Cost: ₹17920.00

Test Case 2: Booking Conflict

Input:

Room Type: "Luxury".

Number of Rooms: 1.

Check-in Date: "2024-12-03".

Check-out Date: "2024-12-08".

Expected Behavior:

1. System checks availability but finds the room booked for overlapping dates.

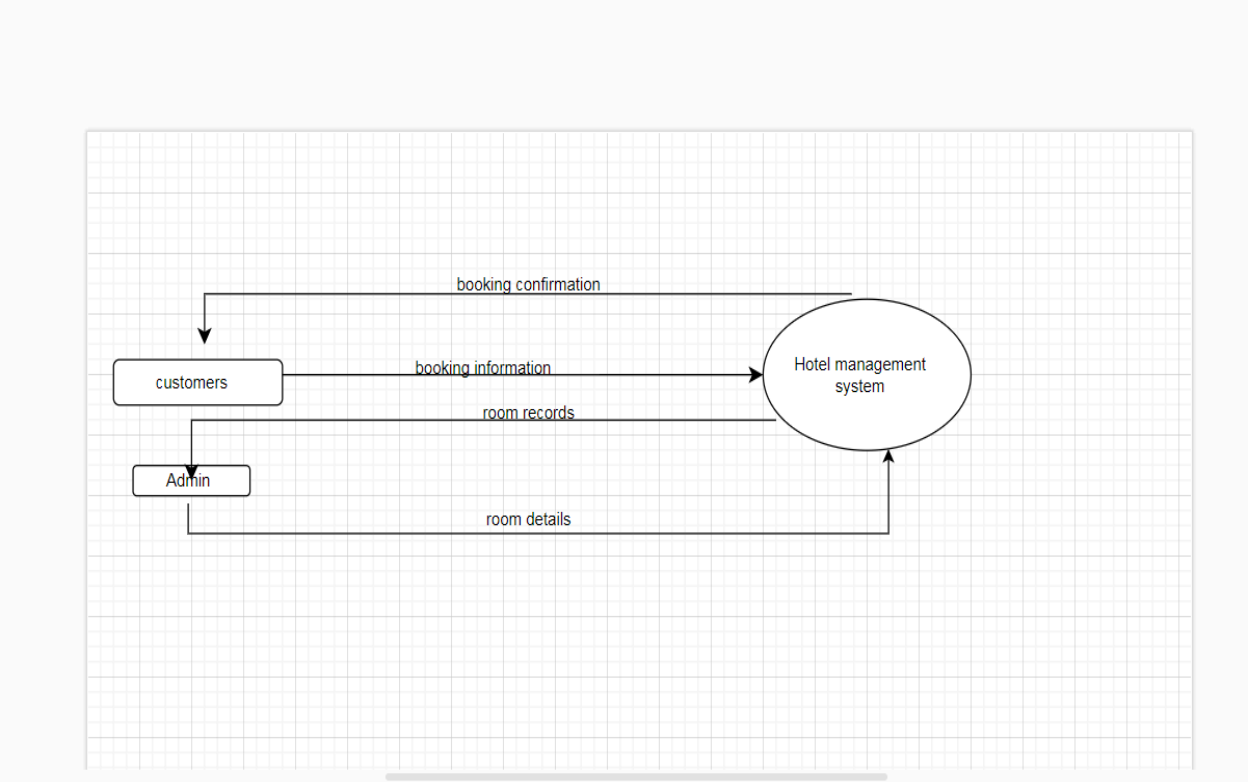
2. Returns a conflict message: "Selected room type is unavailable for the given dates."

10.Analysis Models

10.1 Data Flow Diagram (DFD)

- Level 0 DFD: Context Diagram

The Level 0 DFD represents the overall process as a single system, interacting with external entities. It showcases the primary external data sources and sinks for the system.

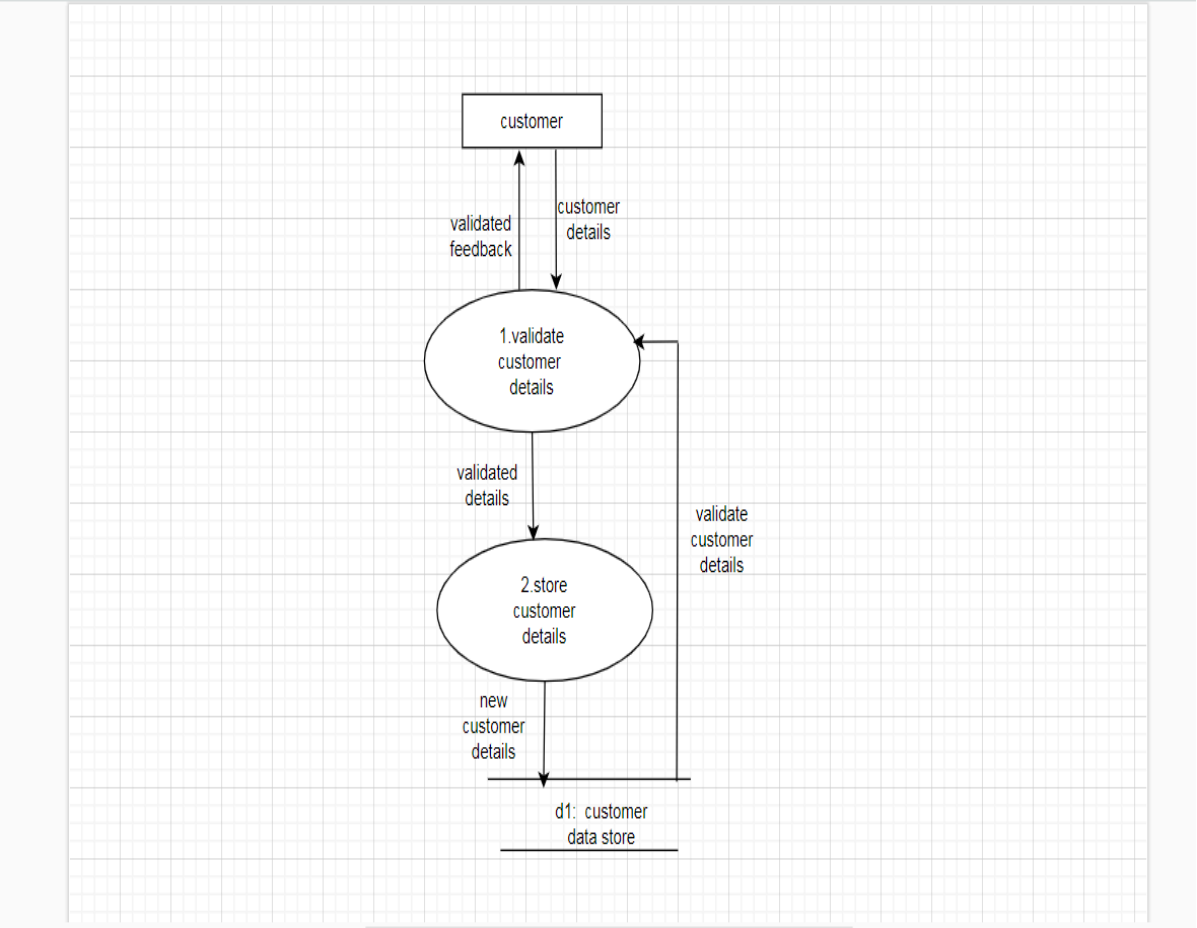


- Level 1 DFD –

  In the Level 1 DFD , the single "Hotel Management System" process is expanded into multiple numbered subprocesses to illustrate its core functionalities.

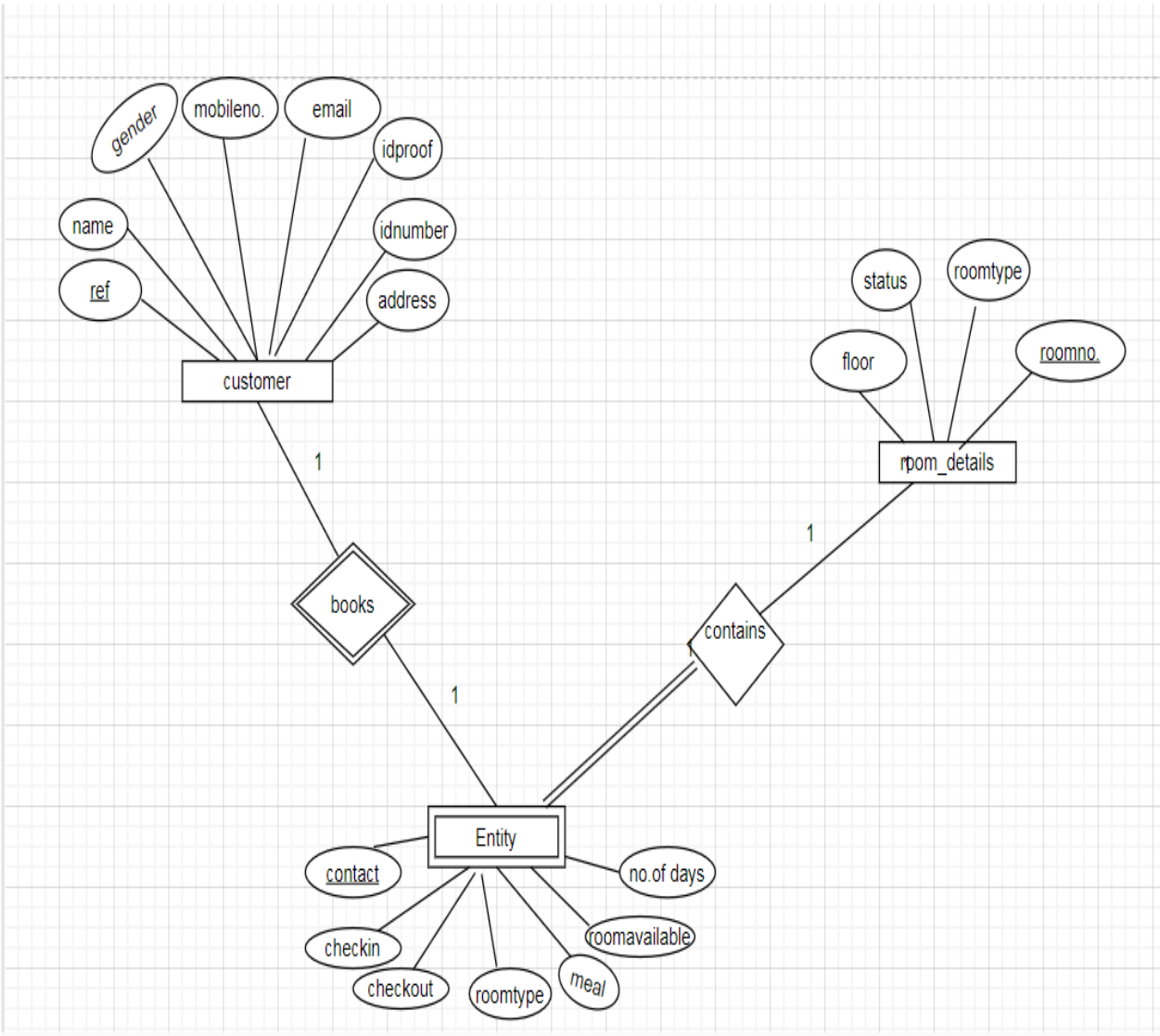
- Level 2 DFD

The Level 2 DFD breaks down individual processes from Level 1 into finer subprocesses. For simplicity, we'll focus on Processes 1.

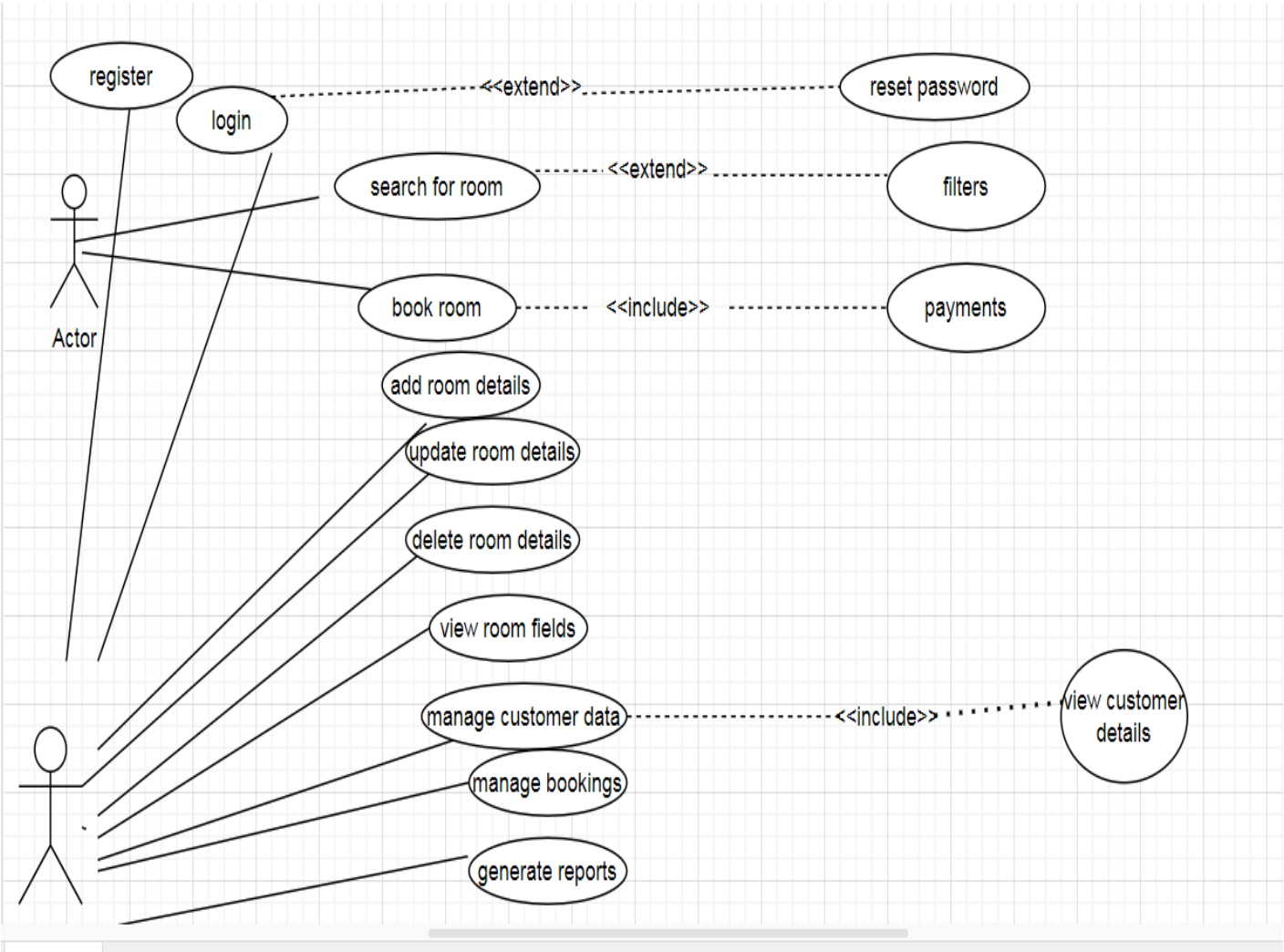


10.2 Database Design

ER Diagram Entities:


gender
mobileno.
email
idproof
name
ref
idnumber
address
customer
status
roomtype
floor
roomno.
rpom_details
1
1
books
contains
1
Entity
contact
no.of days
checkin
roomavailable
checkout
roomtype
meal

10.3 Use case diagram



- use-case diagrams model the behavior of a system.
- used to illustrate the functional requirements of the system and its interaction with external agents (actors).
- A use case diagram gives us a high level view of the system without going into implementation details.

Note – Users can search for rooms and book rooms via admin. It means the user is just providing information to the admin for booking rooms.

11. References

1 IEEE Standard 830-1998 for Software Requirements Specification.

2 Existing manual hotel booking workflows and processes.

3 Python Software Foundation. (2024). Python documentation. Retrieved from https://www.python.org/doc/

4 GeeksforGeeks. (2024). Hotel management system project in Python. Retrieved from https://www.geeksforgeeks.org/

5 W3Schools. (2024). Python Object-Oriented Programming. Retrieved from https://www.w3schools.com/python/

6 Online resources and tutorials related to Python's tkinter library and its use in building GUI applications.

12. Conclusion –

The “Hotel Management System” is a robust and user-friendly solution designed to streamline the management of hotel operations, particularly room bookings. It effectively handles key functionalities such as checking room availability, calculating costs with applicable taxes, and managing customer bookings. This system ensures efficiency by providing accurate booking details, real-time room availability updates, and an organized summary of reservations. By automating these processes, the system minimizes manual errors and enhances customer satisfaction, making it an essential tool for modern hospitality management.

The implementation leverages Python’s capabilities, utilizing object-oriented programming principles for scalability and maintainability. It demonstrates how even a straightforward application can significantly enhance operational efficiency in a hotel environment. With further expansion and integration of additional modules such as customer management, staff management, or online payment systems, this system has the potential to evolve into a comprehensive hotel management suite.