Mini Project Report

On

HOTEL MANAGEMENT SYSTEM

Subject Code: 24CAP-602

Submitted By:

Name: PRIYANSHU

UID: 24MCI10182

Class:MCA (AI&ML)

Section: MAM- 3B

Submitted To:

MISS.SWETA

University Institute of Computing

Chandigarh University,Gharuan,Mohali

UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

TABLE OF CONTENTS

Sr. No.	TOPIC	PAGE NO.
1.	INTRODUCTION	1
2.	BACKGROUND	1,2
3.	OBJECTIVE	3,4,5
4.	TECHNOLOGY USED	4,5
5.	REQUIREMENTS	5,6
6.	SYSTEM DESIGN	6-9
7.	CODE & OUTPUT	9-15
8.	DATABASE USED	15,16
9.	CONCLUSION	16
10	REFERENCES	16

INTRODUCTION

The Hotel Management System (HMS) Mini Project is a simplified version of a comprehensive hotel management software designed to automate and manage various tasks typically handled manually in a hotel. The primary objective of this project is to streamline hotel operations such as guest reservations, room allocation, check-ins and check-outs, billing, and room service management. In essence, this mini project focuses on creating a user-friendly system that allows hotel staff to efficiently handle guest bookings, track room availability, and manage payments in a seamless and organized manner. The system typically includes a database to store and retrieve data related to guests, rooms, payment status, check-in/check-out dates, and other relevant information. With the help of this system, hotel employees can quickly check room availability, assign rooms to customers, and maintain up-to-date guest records, ensuring an error-free experience for both the guests and the staff.

In addition to handling the basic tasks like guest registration and check-out, the HMS mini project can also offer features such as searching for rooms based on room type, date range, or payment status. It allows hotel staff to update guest details and manage room status efficiently. While a full-scale Hotel Management System would include integrations with external systems for accounting, inventory, and housekeeping, the mini project serves as an entry-level project that provides a solid foundation for understanding the core functionalities of hotel operations management.

The user interface is designed to be simple and intuitive, with features like data entry forms for customer details, interactive lists to display available rooms, and buttons for various operations such as adding, deleting, and updating guest records. Typically, this project is built using programming languages like Python with Tkinter for the graphical user interface (GUI), combined with a backend database like SQLite or MySQL for storing data. The Hotel Management System Mini Project is ideal for students or budding developers looking to gain hands-on experience in software development, database management, and application design, all while understanding the day-to-day operational challenges faced by the hospitality industry. The project not only helps improve coding skills but also offers valuable insights into creating real-world applications that can be scaled for use in actual hotel environments.

BACKGROUND

A Hotel Management System (HMS) is a software application designed to manage the operational functions of a hotel. In a traditional hotel, management is often done manually or with legacy software, which can result in inefficiencies and errors. The purpose of building a modern Hotel Management System is to streamline these processes, improve operational efficiency, and provide better services to customers. By automating key tasks such as room booking, guest check-ins, check-outs, reservations, and payment tracking, hotels can operate more effectively, manage resources efficiently, and enhance guest satisfaction.

This project focuses on creating a Hotel Management System using Python and Tkinter, where hotel operations are computerized and managed through an intuitive graphical user interface (GUI). It aims to simplify tasks such as managing customer reservations, tracking room availability, recording check-ins and check-outs, and ensuring proper billing and payments.

OBJECTIVES

The primary objective of this project is to create a fully functional Hotel Management System that can perform the following tasks:

1. **Manage Guest Information:** The system will store and manage guest details such as names, check-in/check-out dates, room numbers, room types, and payment status.
2. **Room Reservation:** It allows staff or customers to reserve rooms for a specific period. The system will track room availability and prevent double bookings.
3. **Customer Check-In/Check-Out:** The system facilitates the check-in process, assigns rooms to customers, and logs their check-out details, including payments.
4. **Data Management:** The system will store customer data in a database (likely SQLite), allowing easy retrieval, updating, and deletion of records. This data can also be searched and displayed.
5. **Room Management**: The system helps in assigning rooms based on customer requirements (e.g., single, double, suite), and it will track which rooms are occupied and which are available.
6. **Payment Handling:** The system tracks payment statuses, ensuring that the hotel can monitor unpaid bills and completed transactions**.**

TECHNOLOGY USED

The Hotel Management System (HMS) project uses a combination of Python, Tkinter, and SQLite. Below is a detailed explanation of the technologies used:

1. Python

- **Role:** Python is the core programming language used to build the logic of the Hotel Management System. It is known for its simplicity, readability, and robust set of libraries, making it an ideal choice for rapid application development.
- **Features:**
    - **Object-Oriented Programming**: The system is designed using Object-Oriented Programming (OOP) principles, with a class-based structure. The

- Hotel class handles the user interface (UI) and the core functionalities of the application.
- **Data Management:** Python's built-in libraries, such as tkinter.messagebox for pop-ups and sqlite3 (presumably in the dbbackend.py file), are used for interacting with databases and handling GUI actions.
- **Cross-platform:** Python can run on various operating systems (Windows, macOS, Linux), making it flexible for deployment**.**

2. Tkinter

- **Role**: Tkinter is the standard Python library for creating Graphical User Interfaces (GUIs). It is used in this project to design the UI components such as buttons, labels, entry fields, frames, listboxes, and scrollbars.
- **Features:**
  - **Widgets:** Various Tkinter widgets like Label, Entry, Button, Listbox, Scrollbar, etc., are used to create interactive UI elements.
  - **Layout Management:** The layout is handled by Tkinter's grid() method, ensuring that the elements resize dynamically as the window is resized (responsive design).

3. SQLite

- **Role:** SQLite is a lightweight database engine used to store and manage data for the Hotel Management System. It is ideal for small-to-medium-scale applications and doesn't require a separate server, making it easy to set up and use**.**
- **Features:**
  - **Database Operations:** SQLite allows performing basic database operations like INSERT, SELECT, UPDATE, and DELETE to manage guest records.
  - **Data Persistence:** The system stores guest information such as room numbers, customer names, check-in/check-out dates, room types, and payment statuses in an SQLite database. This ensures data persistence across different sessions of the application.
  - **dataUpdate(): Updates an existing guest record in the database with new information.**

4. GUI Components and Layout

- **Main Window:** The main window is configured with a size of 1500x800 pixels, which ensures that the UI components are spacious and easy to use.

- **Frames and Labels:** The GUI layout is divided into several sections**:**
  - **Title Frame:** Displays the title of the system in a bold, large font.
  - **Button Frame:** Contains buttons for actions like Add New, Display, Clear, Delete, Search, Update, and Exit.
  - **Data Frame:** Contains two sections — one for guest information input (DataFrameLEFT) and another for displaying guest details (DataFrameRIGHT).
- **Listbox and Scrollbar:** The guestList listbox displays the guest records, and the scrollbar allows easy navigation through large datasets**.**

7. Other Libraries

- **tkinter.messagebox:** Used to display message dialogs such as confirmation alerts when performing tasks like exiting the application or deleting a record**.**
- **StringVar():** Used to hold and update the content of the entry fields, allowing easy interaction with the data displayed in the GUI.

REQUIREMENTS

Hardware Requirements:

1. **Processor (CPU):**
   - A modern processor (Intel i3/i5 or AMD Ryzen 3/5 or equivalent).
   - Minimum: Dual-core processor.
   - Recommended: Quad-core processor or better for faster data processing.
2. **Memory (RAM):**
   - Minimum: 4 GB of RAM.
   - Recommended: 8 GB or more for smoother operation, especially if you have a large number of guests**.**
3. **Storage:**
   - Minimum: 500 MB of free disk space (for storing the SQLite database and application files).
   - Recommended: 1 GB or more free disk space if you're handling a lot of guest data or backups.

4. **Display:**
    - Minimum: 1024x768 resolution.
    - Recommended: Full HD (1920x1080) or better, especially if multiple windows will be open simultaneously**.**
5. **Keyboard and Mouse:**
    - Standard keyboard and mouse for navigation and input.
    - A touchscreen (optional) can be used for more interactive interfaces, especially in kiosk mode.

Software Requirements:

1. **Operating System:**
    - Windows: Windows 7, 8, 10, or 11 (32-bit or 64-bit).
    - macOS: macOS 10.13 (High Sierra) or later.
    - Linux: Any Linux distribution (Ubuntu, Debian, Fedora, etc.).
2. **Python:**
    - Python 3.6 or later. It's recommended to use Python 3.8 or higher for better compatibility with libraries.
    - Python can be downloaded from the official Python website.
3. **Libraries/Frameworks:**
    - **Tkinter:** A standard Python library for GUI development (it comes pre-installed with Python).
    - **SQLite3**: A lightweight, serverless database that is included with Python for data storage.
        - This is used to store guest information, room data, check-in/check-out records, etc.

SYSTEM DESIGN

User Interface

The UI layout consists of several frames and widgets organized in a structured grid:

- **Main Frame**: The root frame that holds the entire UI.
    - Title Frame : Displays the title of the application, "Hotel Management System."
    - Button Frame (ButtonFrame): Holds action buttons for functionalities such as Add New, Display, Clear, Delete, Search, Update, and Exit.
    - Data Frame (DataFrame): Contains two sections:
        - Left Section (DataFrameLEFT): Contains input fields (Labels and Entry widgets) for guest details (Room number, Customer name, etc.).
        - Right Section (DataFrameRIGHT): Contains a Listbox with a scrollbar for displaying guest records.
- **Widgets in DataFrameLEFT:**
    - Labels and Entry fields for entering guest information:
        - Room Number
        - Customer Name
        - Check-In Date
        - Check-Out Date
        - Room Type
        - Payment Status
- **Widgets in DataFrameRIGHT:**
    - Listbox to display the list of guest records.
    - Scrollbar for scrolling through the guest list.
- **Buttons (inside ButtonFrame):**
    - Add New: Adds a new guest record.
    - Display: Displays all guest records in the Listbox.
    - Clear: Clears all entry fields.
    - Delete: Deletes a selected guest record.
    - Search: Searches for guest records based on certain criteria.
    - Update: Updates the selected guest record.

- Exit: Exits the application**.**

B. Database Backend (dbbackend)

The dbbackend module is responsible for interacting with the database. Based on the functionality names, the backend likely handles the following operations:

- addGuestRec(): Adds a new guest record to the database.
- viewData(): Retrieves and returns all guest records for display.
- deleteRec(): Deletes a specific guest record from the database.
- searchData(): Searches the database based on provided criteria (Room Number, Customer Name, etc.).
- dataUpdate(): Updates an existing guest record in the database.

C. System Logic

- **Button Handlers:** Each button on the UI triggers a corresponding function in the system logic. For example, when the "Add New" button is clicked, the addData() function is called, which interacts with the database backend to add a guest record and then updates the Listbox to reflect the new record.
- **Event Handling:**
  - **guestRec(event):** This function is bound to the Listbox widget. When a user selects a record from the Listbox, it populates the Entry fields with the data from that selected record.
  - **displayData():** Displays all guest records in the Listbox by fetching data from the database using the viewData() function**.**

4. Data Flow in the System

1. **User Interaction:**
   - The user interacts with the UI (enters guest details in the input fields and clicks buttons).
2. **Button Actions:**
   - Add New: When clicked, the addData() function is invoked. It sends the data entered by the user to the database via the dbbackend.addGuestRec() function.

- Display: The displayData() function retrieves all guest records from the database (dbbackend.viewData()) and updates the Listbox.
- Search: The searchDatabase() function filters records based on search criteria, invoking dbbackend.searchData().
- Delete: The deleteData() function deletes the selected record via dbbackend.deleteRec().
- Update: The update() function updates the guest record in the database using dbbackend.dataUpdate().

3. **Database Interaction:**
   - The database stores guest records, which include room number, customer name, check-in/out dates, room type, and payment status. These records are retrieved by the application and displayed in the Listbox or updated when needed.

CODE

```
from tkinter import *
import tkinter.messagebox
import dbbackend

class Hotel:

    def __init__(self, root):
        self.root = root
        self.root.title("Hotel Management System")
        self.root.geometry("1500x800+0+0")  # Set the window size for the application
        self.root.config(bg="#f4f4f4")

        # Define StringVar() variables for the fields
        RoomNumber = StringVar()
        CustomerName = StringVar()
        CheckInDate = StringVar()
        CheckOutDate = StringVar()
        RoomType = StringVar()
        PaymentStatus = StringVar()

        # ------------------FUNCTIONS---------------------------------
```

```
    def iExit():
        iExit = tkinter.messagebox.askyesno("Hotel Management System", "Do you want to exit?")
        if iExit > 0:
            root.destroy()

    def clearData():
        self.txtRoomNumber.delete(0, END)
        self.txtCustomerName.delete(0, END)
        self.txtCheckIn.delete(0, END)
        self.txtCheckOut.delete(0, END)
        self.txtRoomType.delete(0, END)
        self.txtPaymentStatus.delete(0, END)

    def addData():
        if len(RoomNumber.get()) != 0:
                    dbbackend.addGuestRec(RoomNumber.get(),  CustomerName.get(),  CheckInDate.get(),
CheckOutDate.get(), RoomType.get(), PaymentStatus.get())
            guestList.delete(0, END)
                        guestList.insert(END,   self.formatRecord(RoomNumber.get(),   CustomerName.get(),
CheckInDate.get(), CheckOutDate.get(), RoomType.get(), PaymentStatus.get()))

    def displayData():
        guestList.delete(0, END)
        for row in dbbackend.viewData():
            guestList.insert(END, self.formatRecord(row[1], row[2], row[3], row[4], row[5], row[6]))

    def guestRec(event):
        global sd
        searchGuest = guestList.curselection()[0]
        sd = guestList.get(searchGuest)

        self.txtRoomNumber.delete(0, END)
        self.txtRoomNumber.insert(END, sd[1])
        self.txtCustomerName.delete(0, END)
        self.txtCustomerName.insert(END, sd[2])
        self.txtCheckIn.delete(0, END)
        self.txtCheckIn.insert(END, sd[3])
        self.txtCheckOut.delete(0, END)
        self.txtCheckOut.insert(END, sd[4])
        self.txtRoomType.delete(0, END)
        self.txtRoomType.insert(END, sd[5])
        self.txtPaymentStatus.delete(0, END)
```

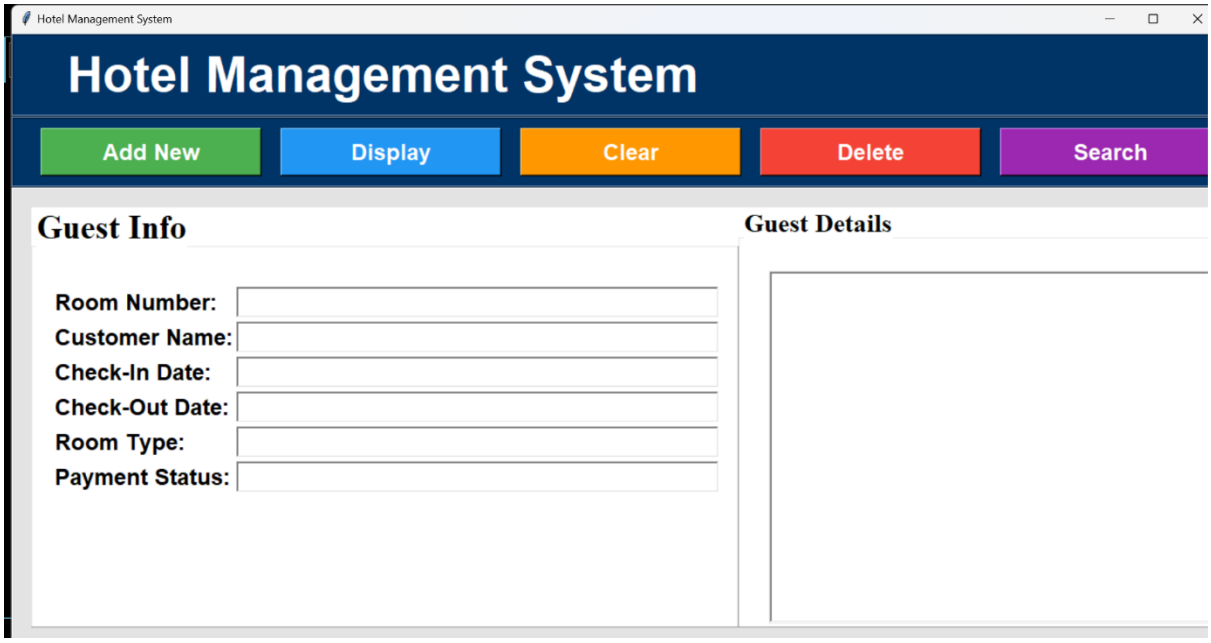
```
        self.txtPaymentStatus.insert(END, sd[6])

    def deleteData():
        if len(RoomNumber.get()) != 0:
            dbbackend.deleteRec(sd[0])
            clearData()
            displayData()

    def searchDatabase():
        guestList.delete(0, END)
        for row in dbbackend.searchData(RoomNumber.get(), CustomerName.get(), CheckInDate.get(),
CheckOutDate.get(), RoomType.get(), PaymentStatus.get()):
            guestList.insert(END, self.formatRecord(row[1], row[2], row[3], row[4], row[5], row[6]))

    def update():
        if len(RoomNumber.get()) != 0:
            dbbackend.dataUpdate(sd[0], RoomNumber.get(), CustomerName.get(), CheckInDate.get(),
CheckOutDate.get(), RoomType.get(), PaymentStatus.get())
            guestList.delete(0, END)
            guestList.insert(END, self.formatRecord(RoomNumber.get(), CustomerName.get(),
CheckInDate.get(), CheckOutDate.get(), RoomType.get(), PaymentStatus.get()))

    # Format record for Listbox display
    def formatRecord(room, customer, checkin, checkout, roomtype, payment):
        return f"Room: {room} | Customer: {customer} | Check-In: {checkin} | Check-Out: {checkout} |
Type: {roomtype} | Status: {payment}"

    # ------------------UI Elements---------------------------------

    MainFrame = Frame(self.root, bg="#f4f4f4")
    MainFrame.grid(row=0, column=0, sticky="nsew")  # Ensure it fills the available space

    TitFrame = Frame(MainFrame, bd=2, padx=54, pady=8, bg="#003366", relief=RIDGE)
    TitFrame.grid(row=0, column=0, sticky="ew")  # Ensures the title frame expands horizontally

        self.lblTit = Label(TitFrame, font=('Arial', 40, 'bold'), text="Hotel Management System",
bg="#003366", fg="white")
    self.lblTit.grid(row=0, column=0)

    ButtonFrame = Frame(MainFrame, bd=2, width=1500, height=70, padx=19, pady=10, bg="#003366",
relief=RIDGE)
    ButtonFrame.grid(row=1, column=0, sticky="ew")  # Ensures the button frame expands horizontally
```

```
        DataFrame = Frame(MainFrame, bd=1, width=1300, height=400, padx=20, pady=20, relief=RIDGE,
bg="#e3e3e3")
        DataFrame.grid(row=2, column=0, sticky="nsew")  # Make sure this frame expands as well

        DataFrameLEFT = LabelFrame(DataFrame, bd=1, width=1000, height=600, padx=20, relief=RIDGE,
bg="white", font=('times new roman', 26, 'bold'), text="Guest Info\n")
        DataFrameLEFT.grid(row=0, column=0, sticky="nsew")  # Left part of the data frame

          DataFrameRIGHT = LabelFrame(DataFrame, bd=1, width=450, height=300, padx=31, pady=3,
relief=RIDGE, bg="white", font=('times new roman', 20, 'bold'), text="Guest Details\n")
        DataFrameRIGHT.grid(row=0, column=1, sticky="nsew")  # Right part of the data frame

        # -------------------Entries----------------------------------

          self.lblRoomNumber = Label(DataFrameLEFT, font=('Arial', 18, 'bold'), text="Room Number:",
padx=2, pady=2, bg="white")
        self.lblRoomNumber.grid(row=0, column=0, sticky=W)
            self.txtRoomNumber = Entry(DataFrameLEFT, font=('Arial', 18), textvariable=RoomNumber,
width=39, bd=2)
        self.txtRoomNumber.grid(row=0, column=1)

         self.lblCustomerName = Label(DataFrameLEFT, font=('Arial', 18, 'bold'), text="Customer Name:",
padx=2, pady=2, bg="white")
        self.lblCustomerName.grid(row=1, column=0, sticky=W)
          self.txtCustomerName = Entry(DataFrameLEFT, font=('Arial', 18), textvariable=CustomerName,
width=39, bd=2)
        self.txtCustomerName.grid(row=1, column=1)

         self.lblCheckIn = Label(DataFrameLEFT, font=('Arial', 18, 'bold'), text="Check-In Date:", padx=2,
pady=2, bg="white")
        self.lblCheckIn.grid(row=2, column=0, sticky=W)
          self.txtCheckIn = Entry(DataFrameLEFT, font=('Arial', 18), textvariable=CheckInDate, width=39,
bd=2)
        self.txtCheckIn.grid(row=2, column=1)

        self.lblCheckOut = Label(DataFrameLEFT, font=('Arial', 18, 'bold'), text="Check-Out Date:", padx=2,
pady=2, bg="white")
        self.lblCheckOut.grid(row=3, column=0, sticky=W)
        self.txtCheckOut = Entry(DataFrameLEFT, font=('Arial', 18), textvariable=CheckOutDate, width=39,
bd=2)
```

```
        self.txtCheckOut.grid(row=3, column=1)

        self.lblRoomType = Label(DataFrameLEFT, font=('Arial', 18, 'bold'), text="Room Type:", padx=2,
pady=2, bg="white")
        self.lblRoomType.grid(row=4, column=0, sticky=W)
        self.txtRoomType = Entry(DataFrameLEFT, font=('Arial', 18), textvariable=RoomType, width=39,
bd=2)
        self.txtRoomType.grid(row=4, column=1)

        self.lblPaymentStatus = Label(DataFrameLEFT, font=('Arial', 18, 'bold'), text="Payment Status:",
padx=2, pady=2, bg="white")
        self.lblPaymentStatus.grid(row=5, column=0, sticky=W)
        self.txtPaymentStatus = Entry(DataFrameLEFT, font=('Arial', 18), textvariable=PaymentStatus,
width=39, bd=2)
        self.txtPaymentStatus.grid(row=5, column=1)

        # -------------------Listbox and Scrollbar----------------------

        scroll = Scrollbar(DataFrameRIGHT)
        guestList = Listbox(DataFrameRIGHT, width=50, height=16, font=('Arial', 14), bd=2)
        scroll.pack(side=RIGHT, fill=Y)
        guestList.pack(side=LEFT, fill=BOTH, expand=True)
        scroll.config(command=guestList.yview)
        guestList.bind("<ButtonRelease-1>", guestRec)

        # -------------------Buttons-----------------------------------

        self.btnAddNew = Button(ButtonFrame, text="Add New", font=('Arial', 18, 'bold'), width=15,
bg="#4CAF50", fg="white", command=addData)
        self.btnAddNew.grid(row=0, column=0, padx=10)

        self.btnDisplay = Button(ButtonFrame, text="Display", font=('Arial', 18, 'bold'), width=15,
bg="#2196F3", fg="white", command=displayData)
        self.btnDisplay.grid(row=0, column=1, padx=10)

        self.btnClear = Button(ButtonFrame, text="Clear", font=('Arial', 18, 'bold'), width=15, bg="#FF9800",
fg="white", command=clearData)
        self.btnClear.grid(row=0, column=2, padx=10)

        self.btnDelete = Button(ButtonFrame, text="Delete", font=('Arial', 18, 'bold'), width=15, bg="#f44336",
fg="white", command=deleteData)
        self.btnDelete.grid(row=0, column=3, padx=10)
```

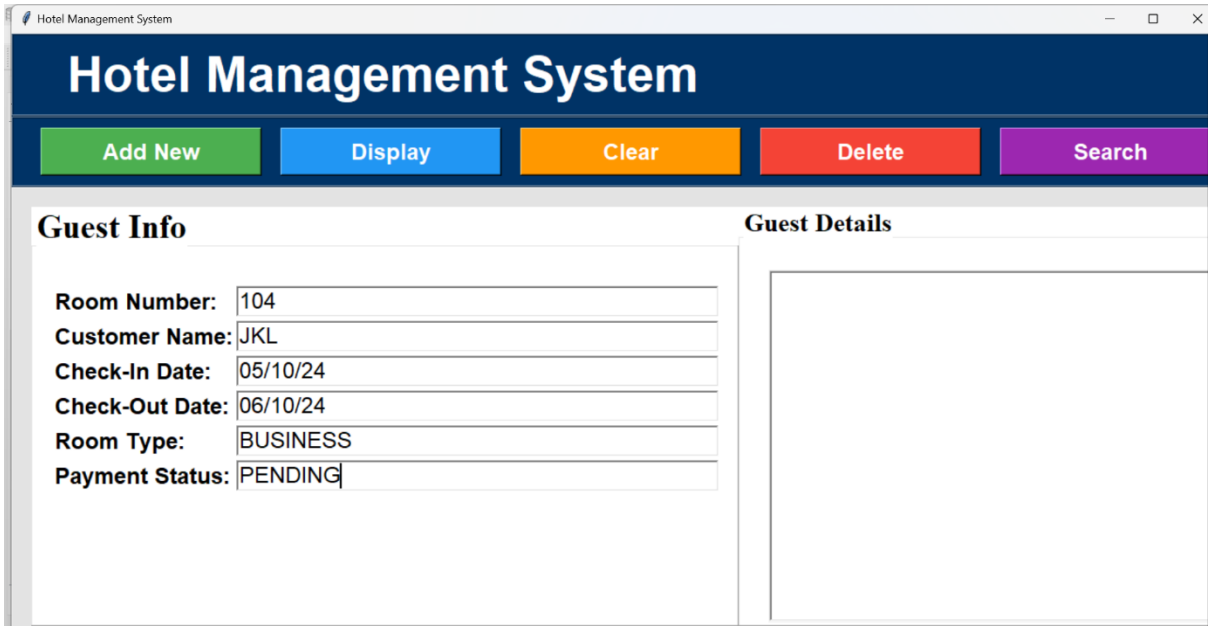
```
        self.btnSearch = Button(ButtonFrame, text="Search", font=('Arial', 18, 'bold'), width=15, bg="#9C27B0", fg="white", command=searchDatabase)
        self.btnSearch.grid(row=0, column=4, padx=10)

        self.btnUpdate = Button(ButtonFrame, text="Update", font=('Arial', 18, 'bold'), width=15, bg="#3F51B5", fg="white", command=update)
        self.btnUpdate.grid(row=0, column=5, padx=10)

        self.btnExit = Button(ButtonFrame, text="Exit", font=('Arial', 18, 'bold'), width=15, bg="#607D8B", fg="white", command=iExit)
        self.btnExit.grid(row=0, column=6, padx=10)

if __name__ == '__main__':
    root = Tk()
    hotel = Hotel(root)
    root.mainloop()
```

OUTPUT



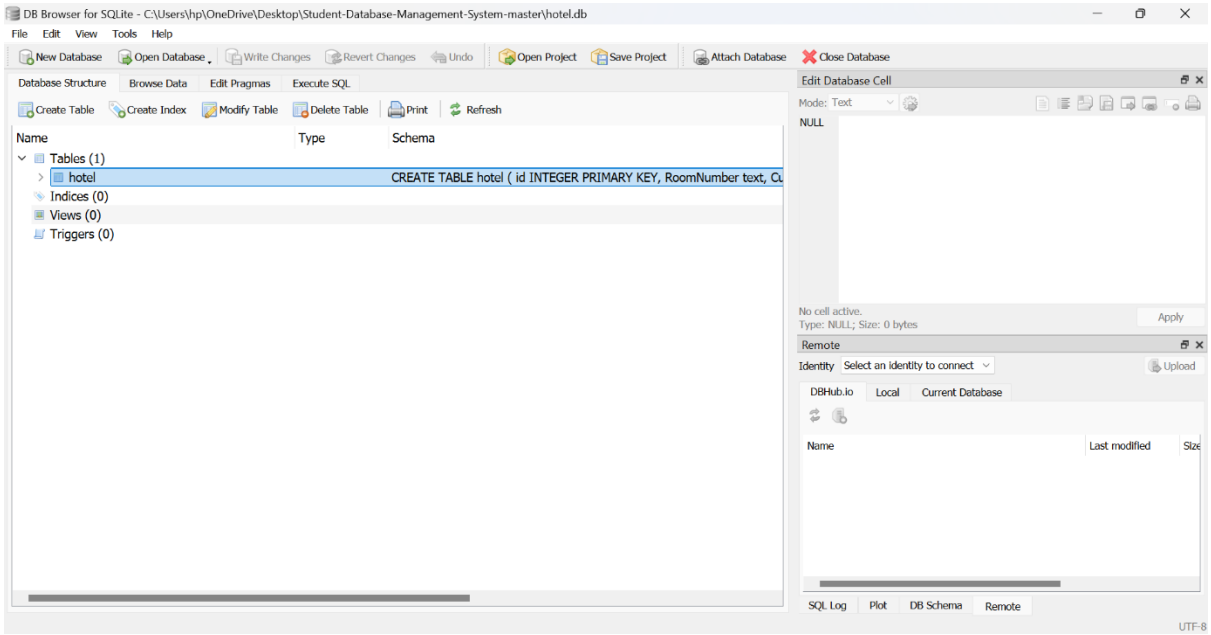
UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University



DATABASE USED

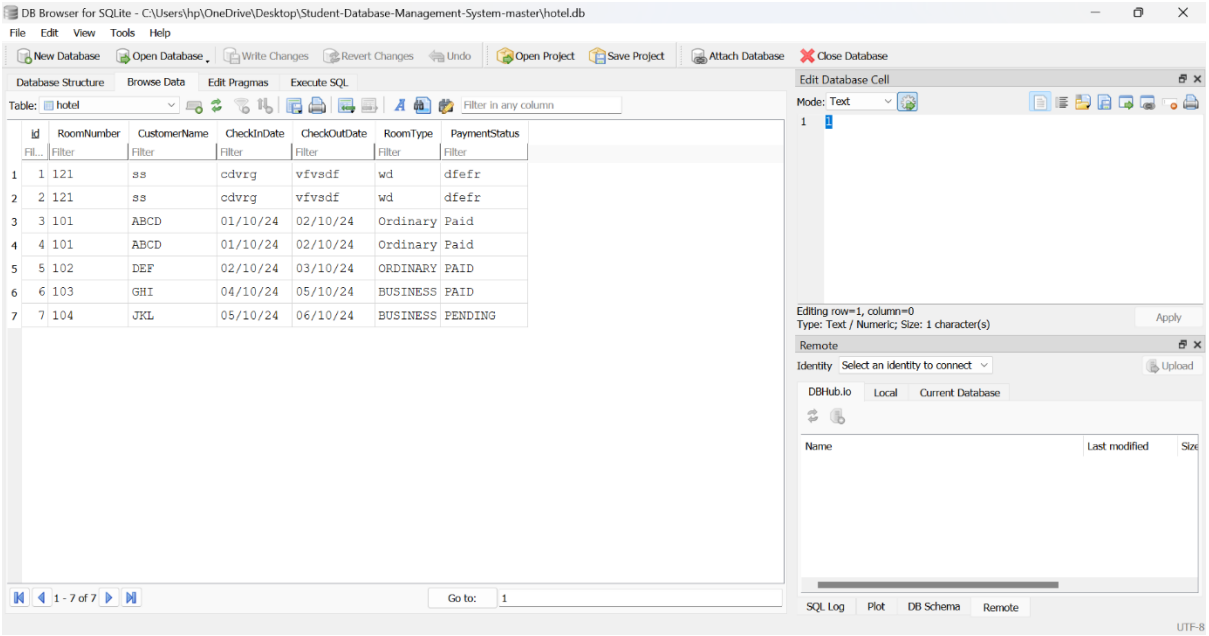
DATABASE STRUCTURE



UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

DATABASE ENTRIES



CONCLUSION

The Hotel Management System project provides a comprehensive and user-friendly solution for managing hotel guest records, enabling efficient operations in a hotel setting. The system integrates a graphical user interface (GUI) built with Tkinter and connects seamlessly to a backend database for storing and retrieving guest information. Through this project, we've demonstrated how a desktop application can automate essential hotel management tasks

REFERENCES

1. Python and Tkinter Programming" by John E. Grayson
2. Python Programming: An Introduction to Computer Science" by John Zelle
3. Official Python Documentation - https://docs.python.org/
4. Tkinter Documentation - https://wiki.python.org/moin/TkInter
5. SQLite Documentation - https://www.sqlite.org/docs.html
6. GeeksforGeeks - https://www.geeksforgeeks.org.