# HOTEL RESERVATION SYSTEM

***Submitted by***

|  |  |
| --- | --- |
| **BALAJEE E** | - **RA2311003020143** |
| **SAI KIRAN S** | - **RA2311003020162** |
| **HARI V** | - **RA2311003020168** |
| **YASHWANTH R** | - **RA2311003020173** |

Under the guidance of

**Dr. V. GOWRI**

**(Assistant Professors, Department of Computer Science and Engineering)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

***of***

**FACULTY OF ENGINEERING AND TECHNOLOGY**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY RAMAPURAM, CHENNAI-600089**

**OCTOBER 2024**

## SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University Under Section 3 of UGC Act, 1956)

# BONAFIDE CERTIFICATE

Certified that the Mini Project titled “HOTEL RESERVATION SYSTEM” is the Bonafide certificate OF BALAJEE E (RA2311003020143), SAI KIRAN S (RA2311003020162), HARI V (RA2311003020168) and YASHWANTH R (RA2311003020173) of II Year CSE submitted for the course 21CSC201J- Data Structures and Algorithms for the Academic Year 2023 – 24 (Odd Semester).

**SIGNATURE**

**Dr. V. Gowri**

Assistant Professor

Department of Computer Science & Engineering **SRM Institute of Science and Technology Ramapuram, Chennai- 600089**

## TABLE OF CONTENT

|  |  |  |
| --- | --- | --- |
| **S No.** | **Title** | **Page No.** |
| 1. | Scope of the project | 4 |
| 2. | Objective | 5 |
| 3. | Problem Statement | 6 |
| 4. | Problem Description | 7 |
| 5. | Implementation | 8 |
| 5.1. | Source Code | 8 |
| 5.2. | Output | 47 |

1. **SCOPE OF THE PROJECT**

The scope of this project is to develop a comprehensive **room reservation system**

that allows users to book, manage, and cancel reservations efficiently.

The system will cater to both **administrative users** (hotel management) and **clients** (guests). It will manage various room types, track room availability in real-time, and provide automated processes for confirming and cancelling reservations.

It will also include a pricing system that adjusts based on the type of room and length of stay. The project covers the design, implementation, and testing of both the front- end and back-end components of the system, integrating a robust database for storing room information and reservations. The system will be capable of handling real-time updates, provide user authentication, and store reservation history for future reference.

## OBJECTIVE

The primary objective of this project is to develop a **scalable, reliable, and user- friendly room reservation system** that can streamline hotel operations and improve customer experience. Specific objectives include:

1. **Simplifying the Reservation Process**: Provide guests with an intuitive, easy-to-use interface that allows them to search for available rooms, check pricing, and make reservations within minutes.
2. **Minimizing Overbooking**: Ensure the system updates room availability in real- time to avoid overbooking and double bookings, ensuring accuracy and customer satisfaction.
3. **Improving Operational Efficiency**: Reduce the need for manual intervention in booking processes, allowing hotel staff to focus on providing quality service.
4. **Enhancing Customer Experience**: Provide seamless booking experiences with features like instant booking confirmation, cancellation options, and secure payment methods.
5. **Increasing Revenue and Occupancy**: Enable hotels to optimize pricing and improve room occupancy rates through real-time tracking and dynamic pricing adjustments.
6. **Data Analytics for Informed Decision-Making**: Equip hotel management with insightful reports and data analytics to help improve decision-making regarding room pricing, marketing campaigns, and operational improvements.
7. **Enabling Multi-channel Accessibility**: Ensure the system is accessible across multiple devices and platforms, catering to tech-savvy guests who prefer to book rooms via mobile devices.

## PROBLEM STATEMENT

Hotels and accommodation providers face significant challenges in managing their room reservation processes. Manual reservation systems or outdated booking platforms can lead to **overbooking, mismanagement of room availability, errors in pricing**, and **customer dissatisfaction**. As the hospitality industry becomes more competitive, hotels require sophisticated reservation systems that not only ensure efficiency but also cater to the evolving needs of digital-savvy customers. Without an effective system in place, hotels struggle to manage their room inventory efficiently, leading to a loss of revenue and a poor guest experience.

## PROBLEM DESCRIPTION

In many hotels, room reservations are either handled manually or through outdated systems that lack the capability to manage complex, real-time booking scenarios. This leads to a number of operational and customer service issues:

1. **Overbooking and Double Booking**: Hotels often face the challenge of overbooking rooms, which occurs when multiple guests book the same room due to lack of real-time updates in availability. This results in inconvenience for guests and the need for hotels to compensate them, often by offering free upgrades or refunds.
2. **Inconsistent Room Availability Updates**: Many manual systems do not reflect real-time room status. For instance, if a room is booked through a third-party platform, this may not immediately be reflected in the hotel's internal system, leading to room availability discrepancies. As a result, staff must manually check and update room availability, which is both time-consuming and prone to error.
3. **Customer Dissatisfaction**: Guests expect a smooth and seamless reservation process. Delays in booking confirmations, mistakes in room assignments, or errors in pricing can lead to frustration and negative reviews. Today’s customers demand features like immediate booking confirmations, easy cancellation policies, and transparent pricing.
4. **Inefficient Use of Hotel Resources**: Managing reservations manually requires significant human resources, especially during peak seasons when hotel occupancy is high. Staff may become overwhelmed with handling booking requests, leading to errors and a decline in service quality. This inefficiency also prevents hotels from allocating resources more effectively
5. **Revenue Loss**: Without an automated system for room reservations, hotels may miss out on opportunities to optimize room pricing and fill available rooms. A lack of dynamic pricing models means that hotels may not be adjusting their prices based on demand, seasonality, or room type, resulting in lost revenue.
6. **Fragmented Customer Data**: Without a centralized system for managing guest information, hotels may struggle to track guest preferences, booking history, and feedback. This fragmentation makes it difficult for hotels to deliver personalized experiences, build customer loyalty, or target marketing campaigns effectively.

The proposed room reservation system addresses these issues by creating a **centralized, automated platform** that integrates all aspects of hotel booking management. By providing real-time updates, reducing manual input, and streamlining the booking process, the system will help hotels increase efficiency, improve customer satisfaction, and ultimately boost revenue.

## IMPLEMENTATION

**Source Code:**

#include <iostream>

#include <string>

#include <conio.h>

#include <fstream>

#include <cstdlib>

#include <algorithm>

#include <sstream>

#include <limits>

#include "Account.hpp"

#include "HRS.hpp"

#include "Account\_setting.hpp"

#include "Sorted\_roomprice.hpp"

#include "ReservationSystem.hpp"

using namespace std;

// Function to display the client menu

void displayClientMenu() {

    cout << "=== Client Menu ===" << endl;

    cout << "[a] Room Reservation" << endl;

    cout << "[b] Display reservation/s" << endl;

    cout << "[c] Account settings" << endl;

    cout << "[d] Log Out" << endl;

    cout << "[e] Exit" << endl;

    cout << "Enter your choice: ";

}

void RoomReserve(){

    cout << "=== Room Reservation ===" << endl;

    cout << "[a] Schedule Room Reservation" << endl;

    cout << "[b] View room types and amenities" << endl;

    cout << "[c] Show available rooms" << endl;

    cout << "[d] Back" << endl;

    cout << "[e] Exit" << endl;

    cout << "Enter your choice: ";

}

void Reservation(HotelReservationSystem& sys, const string& loggedInUsername) {

    char choice;

    do {

        system("cls");

        RoomReserve();

        cin >> choice;

        switch (choice) {

            case 'a':

                sched(loggedInUsername); // Pass the loggedInUsername to sched

                break;

            case 'b':

                char choice;

                system("cls");

                displayRoomInformation();

                while(true){

                    cout << "[a]Back: ";

                    cin >> choice;

                    if(choice == 'a'){

                        break;

                    } else{

                        cout << "Invalid. Try again." << endl;

                    }

                }

                break;

            case 'c':

                int chosenMonth, chosenFromDate;

                cout << "Enter the month number (1-12): ";

                cin >> chosenMonth;

                cout << "Enter the date (1-" << calendar.getMonthDays(chosenMonth, 2024) << "): ";

                cin >> chosenFromDate;

                DisplayRoomAvailability(chosenMonth, chosenFromDate);

                while(true){

                    cout << endl;

                    cout << "[a]Back: ";

                    cin >> choice;

                    if(choice == 'a'){

                        break;

                    } else{

                        cout << "Invalid. Try again." << endl;

                    }

                }

                system("pause");

                break;

            case 'd':

                return;

            case 'e':

                exit(0);

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

    } while(choice != 'd');

}

// Function to handle the main admin menu

void AdminMenu(HotelReservationSystem& sys) {

    Display\_reservation\_admin();

}

void ClientMenu(HotelReservationSystem& sys, const string& loggedInUsername) {

    char choice;

    do {

        system("cls");

        displayClientMenu();

        cin >> choice;

        switch (choice) {

            case 'a':

                Reservation(sys, loggedInUsername); // Pass loggedInUsername to Reservation

                break;

            case 'b':

                Display\_reservation\_client(loggedInUsername);

                break;

            case 'c':

                Account\_setting(sys, loggedInUsername);

                break;

            case 'd':

                cout << "Logging out..." << endl;

                break;

            case 'e':

                cout << "Exiting..." << endl;

                exit(0);

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

        // Clear the input buffer to avoid invalid inputs causing issues in the next loop iteration

        cin.ignore(numeric\_limits<streamsize>::max(), '\n');

    } while (choice != 'd');

}

// Function to login

void showLoginPage(HotelReservationSystem& sys) {

    Account account;

    system("cls");

    cout << "Login System" << endl;

    char choice;

    cout << "[a] Register" << endl;

    cout << "[b] Login" << endl;

    cout << "[c] Exit" << endl;

    cout << "Enter your choice: ";

    cin >> choice;

    while (cin.fail() || (choice != 'a' && choice != 'b' && choice != 'c')) {

        cout << "Invalid choice. Please try again: ";

        cin.clear(); // Clear the error flag

        cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Clear the input buffer

        cin >> choice;

    }

    switch (choice) {

        case 'a':

            account.registerUser();

            break;

        case 'b': {

            string loggedInUsername = account.loginUser();

            if (!loggedInUsername.empty()) {

                if (loggedInUsername == "admin") {

                    cout << "Admin is logged in." << endl;

                    AdminMenu(sys);

                } else {

                    cout << "Logged in successfully." << endl;

                    ClientMenu(sys, loggedInUsername); // Pass loggedInUsername to ClientMenu

                }

            }

            break;

        }

        case 'c':

            cout << "Exiting..." << endl;

            exit(0);

        default:

            cout << endl << "Invalid Input" << endl;

            break;

    }

    // Clear the input buffer

    cin.ignore(numeric\_limits<streamsize>::max(), '\n');

    system("pause");

}

int main() {

    HotelReservationSystem sys;

    loadReservationsFromFile();

    while (true) {

        showLoginPage(sys);

    }

    cin.ignore(numeric\_limits<streamsize>::max(), '\n');

    return 0;

}

#define ACCOUNT\_SETTING\_HPP

#include <iostream>

#include <string>

#include <conio.h>

#include <fstream>

#include <cstdlib>

#include <algorithm>

#include <sstream>

using namespace std;

void Account\_setting(HotelReservationSystem& sys, const string& loggedInUsername) {

    Account settings;

    system("cls");

    cout << "=== Account Settings ===" << endl;

    char choice;

    cout << "[a] User Info" << endl;

    cout << "[b] Change Password" << endl;

    cout << "[c] Update Email" << endl;

    cout << "[d] Update Contact Number" << endl;

    cout << "[e] Back" << endl;

    cout << "[f] Exit" << endl;

    cout << "Enter your choice: ";

    cin >> choice;

    switch (choice) {

        case 'a': {

            string uname = loggedInUsername;

            system("cls");

            if (!loggedInUsername.empty()) {

                // User logged in successfully

                settings.printUserInfo(uname);

            }

            cout << endl;

            cout << "[a]Back     [b]Exit: ";

            cin >> choice;

            if (choice == 'a') {

                break;

            } else if (choice == '0') {

                exit(0);

            }

            break;

        }

        case 'b': {

            string currentPassword;

            string newPassword;

            cout << "Enter current password: ";

            cin >> currentPassword;

            if (!settings.isCurrentPasswordValid(loggedInUsername, currentPassword)) {

                cout << "Incorrect current password." << endl;

                break;

            }

            cout << "Enter new password: ";

            cin >> newPassword;

            settings.changePassword(loggedInUsername, currentPassword, newPassword);

            break;

        }

        case 'c': {

            string currentPassword;

            string newEmail;

            cout << "Enter current password: ";

            cin >> currentPassword;

            if (!settings.isCurrentPasswordValid(loggedInUsername, currentPassword)) {

                cout << "Incorrect current password." << endl;

                break;

            }

            cout << "Enter new email: ";

            cin >> newEmail;

            settings.updateEmail(loggedInUsername, currentPassword, newEmail);

            break;

        }

        case 'd': {

            string currentPassword;

            string newContactNumber;

            cout << "Enter current password: ";

            cin >> currentPassword;

            if (!settings.isCurrentPasswordValid(loggedInUsername, currentPassword)) {

                cout << "Incorrect current password." << endl;

                system("pause");

                break;

            }

            cout << "Enter new contact number: ";

            cin >> newContactNumber;

            settings.updateContactNumber(loggedInUsername, currentPassword, newContactNumber);

            break;

        }

        case 'e':

            return;

        case 'f':

            exit(0);

        default:

            cout << endl << "Invalid Input" << endl;

            break;

    }

}

#define ACCOUNT\_HPP

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <algorithm>

#include <sstream>

#include <Windows.h>

#include <conio.h>

using namespace std;

class Account {

private:

    string username;

    string password;

    string name;

    string email;

    string contactNumber;

    string encryptPassword(const string& password) {

        string encryptedPassword = password;

        // Perform a simple hash by reversing the password

        reverse(encryptedPassword.begin(), encryptedPassword.end());

        return encryptedPassword;

    }

    bool isUserExists(const string& username) {

        ifstream file("credentials");

        string line;

        while (getline(file, line)) {

            if (line.substr(0, line.find(',')) == username) {

                return true;

            }

        }

        return false;

    }

    bool isLoginValid(const string& username, const string& password) {

        ifstream file("credentials");

        string line;

        while (getline(file, line)) {

            size\_t commaPos = line.find(',');

            string storedUsername = line.substr(0, commaPos);

            string storedPassword = line.substr(commaPos + 1, line.find(',', commaPos + 1) - commaPos - 1);

            if (storedUsername == username && storedPassword == encryptPassword(password)) {

                return true;

            }

        }

        return false;

    }

    void createUser(const string& username, const string& password) {

        ofstream file("credentials", ios\_base::app);

        file << username << "," << encryptPassword(password) << "," << name << "," << email << "," << contactNumber << endl;

        cout << "User created successfully." << endl;

    }

public:

    bool isValidEmail(const string& email) {

        // Check for the presence of '@' symbol

        size\_t atPos = email.find('@');

        if (atPos == string::npos)

            return false;

        // Check for the presence of '.' after the '@' symbol

        size\_t dotPos = email.find('.', atPos);

        if (dotPos == string::npos)

            return false;

        // Check if there are at least two characters after the last dot

        if (dotPos >= email.length() - 2)

            return false;

        return true;

    }

    void registerUser() {

        system("pause");

        system("cls");

        cout << "=== Register ===" << endl;

        cout << "Enter name: ";

        cin.ignore();

        getline(cin, name);

        bool validEmail = false;

        while (!validEmail) {

            cout << "Enter email: ";

            cin >> email;

            if (!isValidEmail(email)) {

                cout << "Invalid email format. Please enter a valid email." << endl;

            } else {

                validEmail = true;

            }

        }

        bool validContactNumber = false;

        while (!validContactNumber) {

            cout << "Enter contact number (10 digits): ";

            cin >> contactNumber;

            if (contactNumber.length() == 10) {

                bool allDigits = true;

                for (char c : contactNumber) {

                    if (!isdigit(c)) {

                        allDigits = false;

                        break;

                    }

                }

                if (allDigits) {

                    validContactNumber = true;

                }

            }

            if (!validContactNumber) {

                cout << "Invalid contact number. Please enter a valid 10-digit number." << endl;

            }

        }

        bool usernameExists = false;

        while (!usernameExists) {

            cout << "Enter username: ";

            cin >> username;

            if (isUserExists(username)) {

                cout << "Username already exists. Please try again." << endl;

            } else if (username == "admin") {

                cout << "'admin' is restricted as a username. Please try again." << endl;

            } else {

                cout << "Enter password: ";

                cin >> password;

                createUser(username, password);

                usernameExists = true;

            }

        }

    }

    string getMaskedInput() {

        const char maskChar = '\*';

        string input;

        char ch;

        while ((ch = \_getch()) != '\r') {

            if (ch == '\b') {

                if (!input.empty()) {

                    input.pop\_back();

                    cout << "\b \b";

                }

            } else {

                input.push\_back(ch);

                cout << maskChar;

            }

        }

        cout << endl;

        return input;

    }

    string loginUser() {

        system("cls");

        cout << "=== Log In ===" << endl;

        cout << "Enter username: ";

        cin >> username;

        if (username == "admin") {

            cout << "Enter password: ";

            string password = getMaskedInput();

            if (password == "hrsadmin") {

                return "admin";

            } else {

                cout << "Invalid password." << endl;

                return "";

            }

        }

        if (!isUserExists(username)) {

            cout << "User does not exist." << endl;

            return "";

        } else {

            cout << "Enter password: ";

            string password = getMaskedInput();

            if (isLoginValid(username, password)) {

                return username;

            } else {

                cout << "Invalid username or password." << endl;

                return "";

            }

        }

    }

    bool isCurrentPasswordValid(const string& username, const string& currentPassword) {

        ifstream file("credentials");

        string line;

        while (getline(file, line)) {

            size\_t commaPos = line.find(',');

            string storedUsername = line.substr(0, commaPos);

            string storedPassword = line.substr(commaPos + 1, line.find(',', commaPos + 1) - commaPos - 1);

            if (storedUsername == username && storedPassword == encryptPassword(currentPassword)) {

                return true;

            }

        }

        return false;

    }

    void changePassword(const string& username, const string& currentPassword, const string& newPassword) {

        if (!isCurrentPasswordValid(username, currentPassword)) {

            cout << "Incorrect current password." << endl;

            return;

        }

        // Read the credentials file

        ifstream file("credentials");

        string line;

        stringstream newFileContents;

        // Process each line in the file

        while (getline(file, line)) {

            size\_t commaPos = line.find(',');

            string storedUsername = line.substr(0, commaPos);

            string storedPassword = line.substr(commaPos + 1, line.find(',', commaPos + 1) - commaPos - 1);

            if (storedUsername == username && storedPassword == encryptPassword(currentPassword)) {

                // Found the line with the user's credentials and the correct current password,

                // update the password with the new password

                newFileContents << username << "," << encryptPassword(newPassword) << line.substr(commaPos) << endl;

                cout << "Password changed successfully." << endl;

            } else {

                // Keep the line unchanged

                newFileContents << line << endl;

            }

        }

        // Write the updated contents back to the file

        ofstream outFile("credentials");

        outFile << newFileContents.str();

    }

    void updateEmail(const string& username, const string& currentPassword, const string& newEmail) {

        if (!isCurrentPasswordValid(username, currentPassword)) {

            cout << "Incorrect current password." << endl;

            return;

        }

        // Read the credentials file

        ifstream file("credentials");

        string line;

        stringstream newFileContents;

        // Process each line in the file

        while (getline(file, line)) {

            size\_t commaPos = line.find(',');

            string storedUsername = line.substr(0, commaPos);

            string storedPassword = line.substr(commaPos + 1, line.find(',', commaPos + 1) - commaPos - 1);

            if (storedUsername == username && storedPassword == encryptPassword(currentPassword)) {

                // Found the line with the user's credentials and the correct current password,

                // update the email with the new email

                newFileContents << username << "," << storedPassword << "," << name << "," << newEmail << "," << contactNumber << endl;

                cout << "Email updated successfully." << endl;

            } else {

                // Keep the line unchanged

                newFileContents << line << endl;

            }

        }

        // Write the updated contents back to the file

        ofstream outFile("credentials");

        outFile << newFileContents.str();

    }

    void updateContactNumber(const string& username, const string& currentPassword, const string& newContactNumber) {

        if (!isCurrentPasswordValid(username, currentPassword)) {

            cout << "Incorrect current password." << endl;

            return;

        }

        // Read the credentials file

        ifstream file("credentials");

        string line;

        stringstream newFileContents;

        // Process each line in the file

        while (getline(file, line)) {

            size\_t commaPos = line.find(',');

            string storedUsername = line.substr(0, commaPos);

            string storedPassword = line.substr(commaPos + 1, line.find(',', commaPos + 1) - commaPos - 1);

            if (storedUsername == username && storedPassword == encryptPassword(currentPassword)) {

                // Found the line with the user's credentials and the correct current password,

                // update the contact number with the new contact number

                newFileContents << username << "," << storedPassword << "," << name << "," << email << "," << newContactNumber << endl;

                cout << "Contact number updated successfully." << endl;

            } else {

                // Keep the line unchanged

                newFileContents << line << endl;

            }

        }

        // Write the updated contents back to the file

        ofstream outFile("credentials");

        outFile << newFileContents.str();

    }

    void printUserInfo(const string& username) {

        ifstream file("credentials");

        string line;

        while (getline(file, line)) {

            size\_t commaPos = line.find(',');

            string storedUsername = line.substr(0, commaPos);

            if (storedUsername == username) {

                string storedPassword = line.substr(commaPos + 1, line.find(',', commaPos + 1) - commaPos - 1);

                string storedName = line.substr(line.find(',', commaPos + 1) + 1, line.find(',', line.find(',', commaPos + 1) + 1) - line.find(',', commaPos + 1) - 1);

                string storedEmail = line.substr(line.find(',', line.find(',', commaPos + 1) + 1) + 1, line.find(',', line.find(',', line.find(',', commaPos + 1) + 1) + 1) - line.find(',', line.find(',', commaPos + 1) + 1) - 1);

                string storedContactNumber = line.substr(line.find(',', line.find(',', line.find(',', commaPos + 1) + 1) + 1) + 1);

                cout << "=== User Info ===" << endl << endl;

                cout << "Username: " << username << endl;

                cout << "Name: " << storedName << endl;

                cout << "Email: " << storedEmail << endl;

                cout << "Contact Number: " << storedContactNumber << endl;

                return;

            }

        }

        cout << "User information not found." << endl;

    }

};

#define HRS\_HPP

#include <iostream>

#include <string>

#include <conio.h>

#include <fstream>

#include <cstdlib>

#include <algorithm>

#include <sstream>

using namespace std;

// Structure for client information

struct Client {

    string name;

    string email;

    string contactNo;

    string username;

    string password;

};

// Structure for reservation information

struct Reservation {

    string roomType;

    string referenceNumber;

    int month;

    int fromDate;

    int toDate;

    bool confirmed;

    string loggedInUsername;

};

struct Node {

    Reservation reservation;

    Node\* next;

};

class HotelReservationSystem {

private:

    Node\* head;

public:

    HotelReservationSystem() {

        head = nullptr;

    }

    // Function to add a reservation to the linked list

    void addReservation(Reservation res) {

        Node\* newNode = new Node;

        newNode->reservation = res;

        newNode->next = nullptr;

        if (head == nullptr) {

            head = newNode;

        } else {

            Node\* current = head;

            while (current->next != nullptr) {

                current = current->next;

            }

            current->next = newNode;

        }

    }

};

#define SORTED\_ROOMPRICE\_HPP

#include <iostream>

#include <vector>

#include <algorithm>

struct Room {

    std::string type;

    std::string details;

    int price;

};

bool compareByPrice(const Room& room1, const Room& room2) {

    return room1.price > room2.price;

}

void displayRoomInformation() {

    std::vector<Room> rooms = {

        {"PENTHOUSE",

         "  Room Capacity: 10 or more\n"

         "  All amenities included in a Suite Room\n\n"

         "  \*Included: (only for Penthouse guests)\n"

         "     - Expansive space and luxurious design\n"

         "     - Private balcony\n"

         "     - Personalized gourmet dining experience with a \n"

         "      private chef and customized menu\n"

         "     - Private access to a jacuzzi\n"

         "     - Unlimited access to hotel's sport facility and \n"

         "      hotel's recreational facility\n"

         "     - Private butler service for the duration of the stay",

         35000},

         {"EXECUTIVE ROOM",

         "  Room Capacity: 6\n"

         "  All amenities included in a Deluxe Room\n"

         "  Additional Single bedsize (1)\n\n"

         "  \*Included: (only for Executive Room guests)\n"

         "     - Complimentary daily breakfast and exclusive access \n"

         "      to an executive lounge with snacks and beverages\n"

         "     - Access to the hotel's business center and meeting rooms\n"

         "     - Priority access to the hotel's recreational facilities \n"

         "      limited to spa and game room",

         20000},

         {"SUITE ROOM",

         "  Room Capacity: 4\n"

         "  All amenities included in a Deluxe Room\n"

         "  Double bedsize (2)\n"

         "  Separate living room\n"

         "  Kitchenette / fully-equipped kitchen\n\n"

         "  \*Included: (only for Suite Room guests)\n"

         "     - Personalized in-room dining experience with a private chef\n"

         "     - Welcome fruit basket and a bottle of wine upon arrival\n"

         "     - Private yoga session\n"

         "     - Access to the hotel's sports facility limited to bowling \n"

         "      and billiard",

         12000},

         {"DELUXE ROOM",

         "  Room Capacity: 4\n"

         "  All amenities included in a Standard Room\n"

         "  Mini-refrigerator\n"

         "  California King bedsize (1)\n"

         "  Single bedsize (1)\n"

         "  Upgraded furnishings and decor\n\n"

         "  \*Included: (only for Deluxe Room guests)\n"

         "     - Option for a gourmet dining experience at an on-site \n"

         "      restaurant\n"

         "     - Complimentary access to the hotel's fitness center\n"

         "     - Cooking class with a professional chef",

         9000},

         {"STANDARD ROOM",

         "  Room Capacity: 1 or 2\n"

         "  Single bedsize (2)\n"

         "  Private bathroom with shower\n"

         "  Television\n"

         "  Wi-Fi\n"

         "  Air condition\n\n"

         "  \*Included: (only for Standard Room guests)\n"

         "     - Option for in-room dining with a variety of meal \n"

         "      choices available",

         5000}

    };

    std::sort(rooms.begin(), rooms.end(), compareByPrice);

    std::cout << "Room Information (Highest to Lowest Price):" << std::endl;

    for (const Room& room : rooms) {

        std::cout << std::endl << "======================================================================" << std::endl;

        std::cout << "Price: " << room.price << " Rs/night" << std::endl;

        std::cout << "Type: " << room.type << std::endl << std::endl;

        std::cout << room.details << std::endl << std::endl;

        std::cout << "======================================================================" << std::endl;

        std::cout << std::endl;

    }

}

#define RESERVATIONSYSTEM\_HPP

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <fstream>

#include <sstream>

#include <string>

#include <algorithm>

using namespace std;

void confirmSelectedReservation();

void Roomreserve(int chosenMonth, int chosenFromDate, int chosenToDate);

void confirmSelectedReservation();

void cancelReservation();

void sched(const string& loggedInUsername);

void displayRoomInformation();

void DisplayRoomAvailability(int chosenMonth, int chosenFromDate) ;

string generateReferenceNumber();

const int NUM\_MONTHS = 12;

const int NUM\_DAYS\_PER\_WEEK = 7;

class Calendar {

private:

    string monthNames[NUM\_MONTHS] = {

        "January", "February", "March", "April", "May", "June",

        "July", "August", "September", "October", "November", "December"

    };

    int startingDays[NUM\_MONTHS] = {

        1, 4, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0 // Starting days for January to December 2024

    };

public:

    Calendar() {}

    void displayMonth(int month, int year, int startingDay) const {

        string monthName = monthNames[month - 1];

        cout << endl << setw(20) << monthName << " " << year << endl;

        cout << "-----------------------------" << endl;

        cout << "Sun Mon Tue Wed Thu Fri Sat" << endl;

        int numDaysInMonth;

        if (month == 2) {

            if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)

                numDaysInMonth = 29;

            else

                numDaysInMonth = 28;

        } else if (month == 4 || month == 6 || month == 9 || month == 11) {

            numDaysInMonth = 30;

        } else {

            numDaysInMonth = 31;

        }

        int day = 1;

        int currentDayOfWeek = startingDay;

        cout << setw(currentDayOfWeek \* 4) << "";

        while (day <= numDaysInMonth) {

            cout << setw(3) << day << " ";

            day++;

            currentDayOfWeek++;

            if (currentDayOfWeek == NUM\_DAYS\_PER\_WEEK) {

                cout << endl;

                currentDayOfWeek = 0;

            }

        }

        cout << endl;

    }

    int getMonthDays(int month, int year) const {

        if (month == 2) {

            if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)

                return 29;

            else

                return 28;

        } else if (month == 4 || month == 6 || month == 9 || month == 11) {

            return 30;

        } else {

            return 31;

        }

    }

    string getMonthName(int month) const {

        return monthNames[month - 1];

    }

};

struct RoomAvailability {

    int count;

    vector<int> availability;

    RoomAvailability(int c, int days) : count(c), availability(days, c) {}

};

struct RoomType {

    string type;

    vector<RoomAvailability> availabilityByMonth;

    int price;

    RoomType(string t, int c, int days, int p) : type(t), availabilityByMonth(NUM\_MONTHS, RoomAvailability(c, days)), price(p) {}

};

Calendar calendar;

vector<RoomType> roomTypes = {

    {"Standard Room", 15, calendar.getMonthDays(1, 2024), 5000},

    {"Deluxe Room", 10, calendar.getMonthDays(1, 2024), 9000},

    {"Suite Room", 10, calendar.getMonthDays(1, 2024), 12000},

    {"Executive Room", 7, calendar.getMonthDays(1, 2024), 20000},

    {"Penthouse", 3, calendar.getMonthDays(1, 2024), 35000}

};

vector<Reservation> reservations;

void Roomreserve(int chosenMonth, int chosenFromDate, int chosenToDate, const string& loggedInUsername) {

    string roomreserve\_opt;

    int roomIndex = -1;

    while (true) {

        cout << endl;

        cout << "Please select your preferred room type:\n";

        for (int i = 0; i < roomTypes.size(); i++) {

            cout << "[" << static\_cast<char>('A' + i) << "] " << roomTypes[i].type << endl;

        }

        cout << "\n[" << static\_cast<char>('A' + roomTypes.size()) << "] Back\n";

        cout << "[" << static\_cast<char>('A' + roomTypes.size() + 1) << "] Exit\n";

        cout << "Enter your preferred room: ";

        cin >> roomreserve\_opt;

        // Convert the input to uppercase

        transform(roomreserve\_opt.begin(), roomreserve\_opt.end(), roomreserve\_opt.begin(), ::toupper);

        switch (roomreserve\_opt[0]) {

            case 'A':

            case 'B':

            case 'C':

            case 'D':

            case 'E':

                roomIndex = roomreserve\_opt[0] - 'A'; // Assigning value to roomIndex

                break;

            case 'F':

                return;

            case 'G':

                cout << "Thank you for using the Reservation System. Goodbye!\n";

                exit(0);

            default:

                cout << "Invalid entry. Please try again.\n";

                break; // Loop to re-ask for valid input

        }

        if (roomIndex != -1)

            break; // Exit the loop if a valid room selection is made

    }

    if (roomTypes[roomIndex].availabilityByMonth[chosenMonth - 1].availability[chosenFromDate - 1] > 0) {

        string referenceNumber = generateReferenceNumber();

        reservations.push\_back({roomTypes[roomIndex].type, referenceNumber, chosenMonth, chosenFromDate, chosenToDate, false, loggedInUsername});

        // Store the reservation details in unavailable\_date.txt

        ofstream outFile("reservations.txt");

    if (outFile.is\_open()) {

        for (const Reservation& reservation : reservations) {

            outFile << reservation.roomType << "," << reservation.referenceNumber << ","

                    << reservation.month << "," << reservation.fromDate << ","

                    << reservation.toDate << "," << reservation.confirmed << "," << reservation.loggedInUsername << endl;

        }

        outFile.close();

        cout << "Reservations saved successfully." << endl;

    } else {

        cout << "Unable to open the file for writing." << endl;

    }

        // Calculate the total amount based on the price, the duration of the stay, and the priceMultiplier

        int numDays = chosenToDate - chosenFromDate;

        if (numDays == 0) {

            numDays = 1;

        }

        int totalAmount = roomTypes[roomIndex].price \* numDays;

        cout << "\nRoom reserved successfully. Reference number: " << referenceNumber << endl;

        cout << "Total amount: Rs " << totalAmount << ".00" << endl;

        system("pause");

    } else {

        cout << "Sorry, no rooms of this type are available.\n";

    }

}

// Function to load reservations from a file

void loadReservationsFromFile() {

    ifstream inFile("reservations.txt");

    if (inFile.is\_open()) {

        reservations.clear();

        string line;

        while (getline(inFile, line)) {

            //std::cout << line << std::endl;

            istringstream iss(line);

            string roomType, referenceNumber, loggedInUsername, confirmedStr;

            int month, fromDate, toDate;

            bool confirmed;

            if (getline(iss, roomType, ',') &&

                getline(iss, referenceNumber, ',') &&

                (iss >> month) && (iss.ignore()) &&

                (iss >> fromDate) && (iss.ignore()) &&

                (iss >> toDate) && (iss.ignore()) &&

                getline(iss, confirmedStr, ',') &&

                getline(iss, loggedInUsername)) {

                confirmed = (confirmedStr == "1");

                reservations.push\_back({roomType, referenceNumber, month, fromDate, toDate, confirmed, loggedInUsername});

            }

        }

        inFile.close();

    }

}

string generateReferenceNumber() {

    string referenceNumber;

    static const char alphanumeric[] =

        "0123456789"

        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

        "abcdefghijklmnopqrstuvwxyz";

    const int alphanumericLength = sizeof(alphanumeric) - 1;

    srand(time(0));

    for (int i = 0; i < 6; i++) {

        referenceNumber += alphanumeric[rand() % alphanumericLength];

    }

    return referenceNumber;

}

// Global variable to hold the chosen month

int chosenMonth, chosenFromDate, chosenToDate;

vector<vector<RoomType>> roomTypesByMonth(NUM\_MONTHS, vector<RoomType>());

void initializeRoomAvailability(int chosenMonth) {

    // Check if the roomTypes for the chosen month are already initialized

    if (roomTypesByMonth[chosenMonth - 1].empty()) {

        // Initialize the availability vectors for each room type for the chosen month

        for (RoomType& roomType : roomTypes) {

            roomType.availabilityByMonth[chosenMonth - 1].availability = vector<int>(calendar.getMonthDays(chosenMonth, 2024), roomType.availabilityByMonth[chosenMonth - 1].count);

        }

        // Add the roomTypes for the chosen month to the nested vector

        roomTypesByMonth[chosenMonth - 1] = roomTypes;

    } else {

        // If the roomTypes for the chosen month are already initialized,

        // copy them back to the original roomTypes vector to avoid duplication.

        roomTypes = roomTypesByMonth[chosenMonth - 1];

    }

}

void confirmSelectedReservation() {

    string referenceNumber;

    cout << "Enter the reference number of the reservation to confirm: ";

    cin >> referenceNumber;

    bool reservationFound = false;

    for (Reservation& reservation : reservations) {

        if (reservation.referenceNumber == referenceNumber) {

            reservationFound = true;

            // Check if the reservation is already confirmed

            if (reservation.confirmed) {

                cout << "This reservation is already confirmed.\n";

                return;

            }

            // Find the room type index

            int roomIndex = -1;

            for (int i = 0; i < roomTypes.size(); i++) {

                if (roomTypes[i].type == reservation.roomType) {

                    roomIndex = i;

                    break;

                }

            }

            if (roomIndex == -1) {

                cout << "Invalid room type for the reservation.\n";

                return;

            }

            // Call DisplayRoomAvailability to check the number of available rooms

            int chosenMonth = reservation.month;

            int chosenFromDate = reservation.fromDate;

            //DisplayRoomAvailability(chosenMonth, chosenFromDate);

            // Check if the rooms are available for the reservation dates

            int chosenToDate = reservation.toDate;

            for (int i = chosenFromDate; i <= chosenToDate; i++) {

                if (roomTypes[roomIndex].availabilityByMonth[chosenMonth - 1].availability[i - 1] <= 0) {

                    cout << "Sorry, the rooms are not available for the reservation dates.\n";

                    return;

                }

            }

            // Mark the reservation as confirmed

            reservation.confirmed = true;

            // Adjust room availability for the chosen date range of the selected room type

            for (int i = chosenFromDate; i <= chosenToDate; i++) {

                roomTypes[roomIndex].availabilityByMonth[chosenMonth - 1].availability[i - 1]--;

            }

            cout << "Reservation confirmed successfully.\n";

            return;

        }

    }

    if (!reservationFound) {

        cout << "Invalid reference number. Reservation not found.\n";

    }

}

void deleteReservation() {

    string referenceNumber;

    cout << "Enter the reference number of the reservation to cancel: ";

    cin >> referenceNumber;

    bool reservationFound = false;

    for (int i = 0; i < reservations.size(); i++) {

        if (reservations[i].referenceNumber == referenceNumber) {

            reservationFound = true;

            reservations.erase(reservations.begin() + i);

            cout << "Reservation successfully cancelled.\n";

            break;

        }

    }

    if (!reservationFound) {

        cout << "Invalid reference number. Reservation not found.\n";

    }

}

void Display\_reservation\_admin() {

    string display\_opt;

    do{

    system("cls");

    cout << "=== Manage Reservation Schedule ===" << endl << endl;

    if(reservations.size() == 0) {

        cout << "No reservations found." << endl;

    }

    for (int i = 0; i < reservations.size(); i++) {

        Reservation reservation = reservations[i];

        cout << "[" << (i + 1) << "] " << "[Username: " << reservation.loggedInUsername << "] " << "[Room Type: " << reservation.roomType << "] " << " (Date booked: " << reservation.month << "/" << reservation.fromDate << "-" << reservation.toDate << ", Reference number: " << reservation.referenceNumber << ")";

        cout << "  Status: " << (reservation.confirmed ? "CONFIRMED" : "PENDING") << endl;

    }

    cout << "\n[a] Confirm reservation";

    cout << "\n[b] Delete reservation\n";

    cout << "[c] Show Available Rooms\n";

    cout << "[d] Logout\n";

    cout << "[e] Exit\n";

    cout << "Enter your choice: ";

    cin >> display\_opt;

    cout << endl;

    // Convert the input to uppercase

    transform(display\_opt.begin(), display\_opt.end(), display\_opt.begin(), ::toupper);

    switch (display\_opt[0]) {

        case 'A':

            confirmSelectedReservation();

            system("pause");

            break;

        case 'B':

            deleteReservation();

            system("pause");

            break;

        case 'C':

                char choice;

                int chosenMonth, chosenFromDate;

                cout << "Enter the month number (1-12): ";

                cin >> chosenMonth;

                cout << "Enter the date (1-" << calendar.getMonthDays(chosenMonth, 2024) << "): ";

                cin >> chosenFromDate;

                DisplayRoomAvailability(chosenMonth, chosenFromDate);

                while(true){

                    cout << endl;

                    cout << "[a]Back: ";

                    cin >> choice;

                    if(choice == 'a'){

                        break;

                    } else{

                        cout << "Invalid. Try again." << endl;

                    }

                }

                system("pause");

                break;

        case 'D':

            cout << "Logging out..." << endl;

            break;

        case 'E':

            cout << "Thank you for using the Reservation System. Goodbye!\n";

            exit(0);

        default:

            cout << "Invalid entry.\n";

    }

    } while (display\_opt[0] != 'D');

}

void Display\_reservation\_client(const string& loggedInUsername) {

    system("cls");

    cout << "\nMY RESERVATIONS:\n";

    int reservationCount = 0;

    for (int i = 0; i < reservations.size(); i++) {

        Reservation reservation = reservations[i];

        if (reservation.loggedInUsername == loggedInUsername) {

            reservationCount++;

            cout << "[" << reservationCount << "] " << reservation.roomType << " (Date booked: " << reservation.month << "/" << reservation.fromDate << "-" << reservation.toDate << ", Reference number: " << reservation.referenceNumber << ")";

            cout << "  Status: " << (reservation.confirmed ? "CONFIRMED" : "PENDING") << endl;

        }

    }

    if (reservationCount == 0) {

        cout << "You have no reservations." << endl;

    }

    string display\_opt;

    cout << "\n[a] Cancel reservation\n";

    cout << "[b] Back\n";

    cout << "[c] Exit\n";

    cout << "Enter your choice: ";

    cin >> display\_opt;

    cout << endl;

    // Convert the input to uppercase

    transform(display\_opt.begin(), display\_opt.end(), display\_opt.begin(), ::toupper);

    switch (display\_opt[0]) {

        case 'A':

            deleteReservation();

            system("pause");

            break;

        case 'B':

            return;

        case 'C':

            cout << "Thank you for using the Reservation System. Goodbye!\n";

            exit(0);

        default:

            cout << "Invalid entry.\n";

    }

    // Clear the input buffer

    cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

void sched(const string& loggedInUsername) {

    Calendar calendar;

    int chosenFromDate, chosenToDate;

    while (true) {

        cout << "Enter the month number (1-12): ";

        cin >> chosenMonth;

        if (cin.fail() || chosenMonth < 1 || chosenMonth > NUM\_MONTHS) {

            cout << "Invalid month number. Please try again." << endl;

            cin.clear(); // Clear the error flag

            cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Clear the input buffer

        } else {

            break;

        }

    }

    int numDays = calendar.getMonthDays(chosenMonth, 2024);

    // Display the calendar for the chosen month

    calendar.displayMonth(chosenMonth, 2024, 1);

    while (true) {

        cout << "Enter the starting date (1-" << numDays << "): ";

        cin >> chosenFromDate;

        if (chosenFromDate < 1 || chosenFromDate > numDays) {

            cout << "Invalid date." << endl;

        } else {

            break;

        }

    }

    while (true) {

        cout << "Enter the ending date (1-" << numDays << "): ";

        cin >> chosenToDate;

        if (chosenToDate <= chosenFromDate || chosenToDate > numDays) {

            cout << "Invalid date." << endl;

        } else {

            break;

        }

    }

    system("cls");

    cout << "\nROOM AVAILABILITY on " << calendar.getMonthName(chosenMonth) << " " << chosenFromDate << ":\n";

    cout << "Room Type\tAvailable Rooms\n";

    for (const RoomType& roomType : roomTypes) {

        int availableRooms = roomType.availabilityByMonth[chosenMonth - 1].availability[chosenFromDate - 1];

        cout << roomType.type << "\t\t" << availableRooms << endl;

    }

    initializeRoomAvailability(chosenMonth); // Initialize the room availability vector

    Roomreserve(chosenMonth, chosenFromDate, chosenToDate, loggedInUsername);

}

void DisplayRoomAvailability(int chosenMonth, int chosenFromDate) {

    if (chosenMonth < 1 || chosenMonth > NUM\_MONTHS) {

        cout << "Invalid month number." << endl;

        return;

    }

    int numDays = calendar.getMonthDays(chosenMonth, 2024);

    cout << endl << "Room Type\tAvailable Rooms\n";

    for (const RoomType& roomType : roomTypes) {

        int availableRooms = roomType.availabilityByMonth[chosenMonth - 1].availability[chosenFromDate - 1];

        cout << roomType.type << "\t\t" << availableRooms << endl;

    }

}

vector<RoomType> getAvailableRooms(int chosenMonth, int chosenFromDate) {

    vector<RoomType> availableRooms = roomTypesByMonth[chosenMonth - 1];

    // Adjust room availability based on reservations for the chosen date

    for (const Reservation& reservation : reservations) {

        if (reservation.month == chosenMonth && reservation.fromDate <= chosenFromDate && reservation.toDate >= chosenFromDate) {

            for (RoomType& roomType : availableRooms) {

                if (roomType.type == reservation.roomType) {

                    roomType.availabilityByMonth[chosenMonth - 1].availability[chosenFromDate - 1]--;

                }

            }

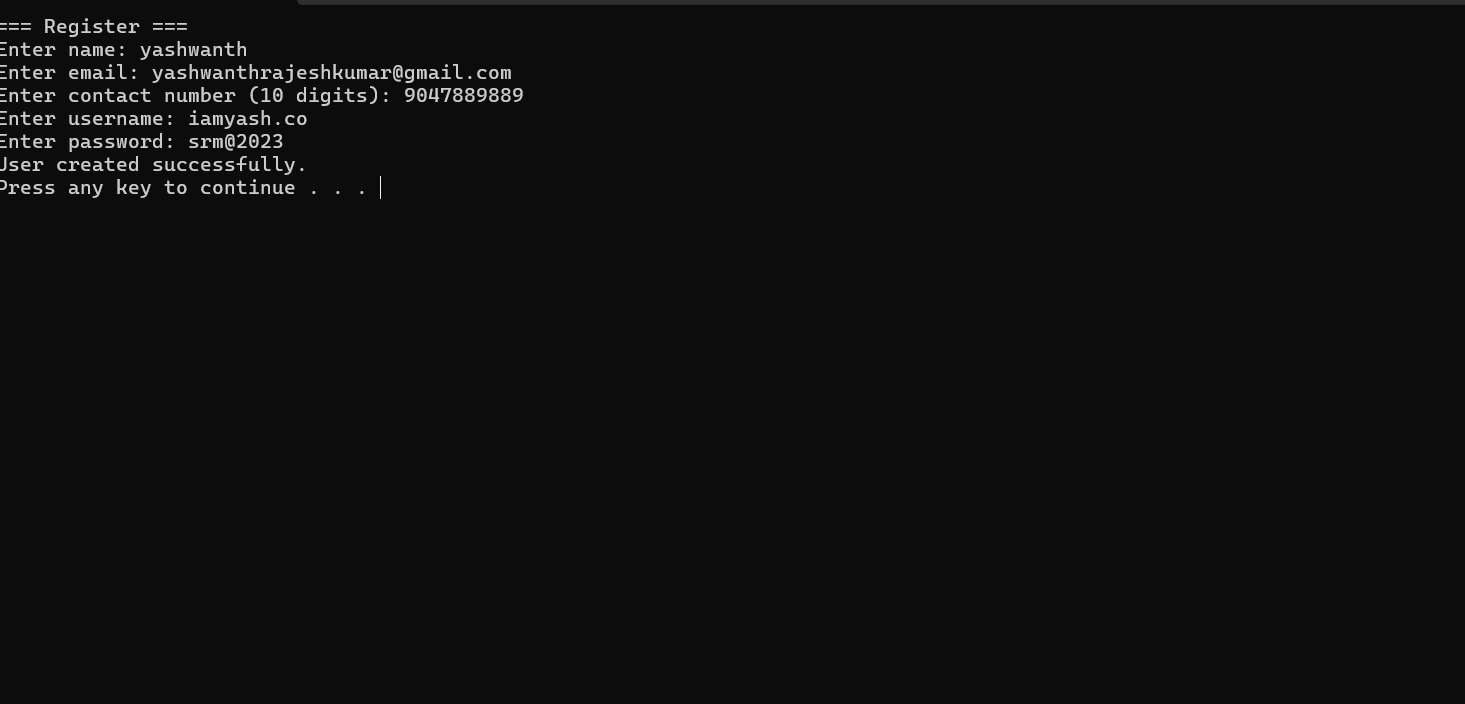
        }

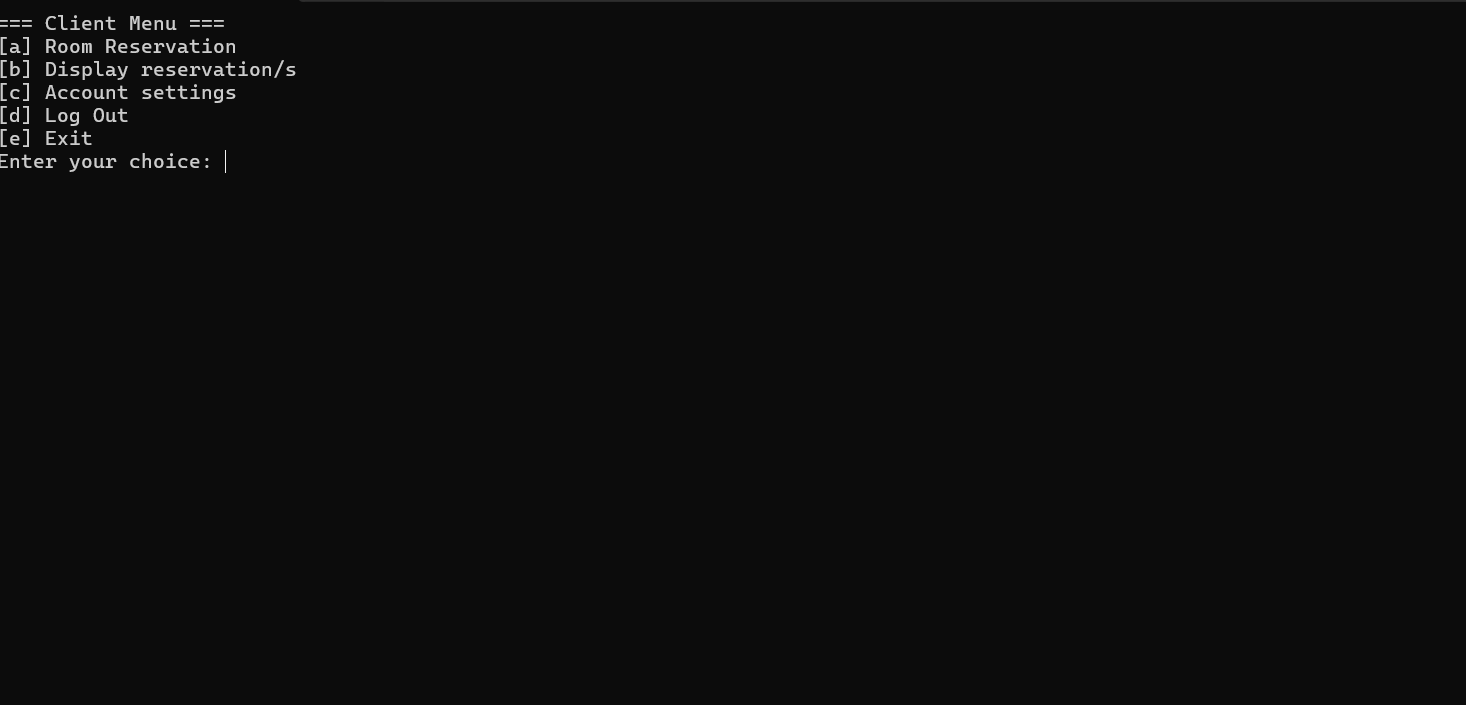
    }

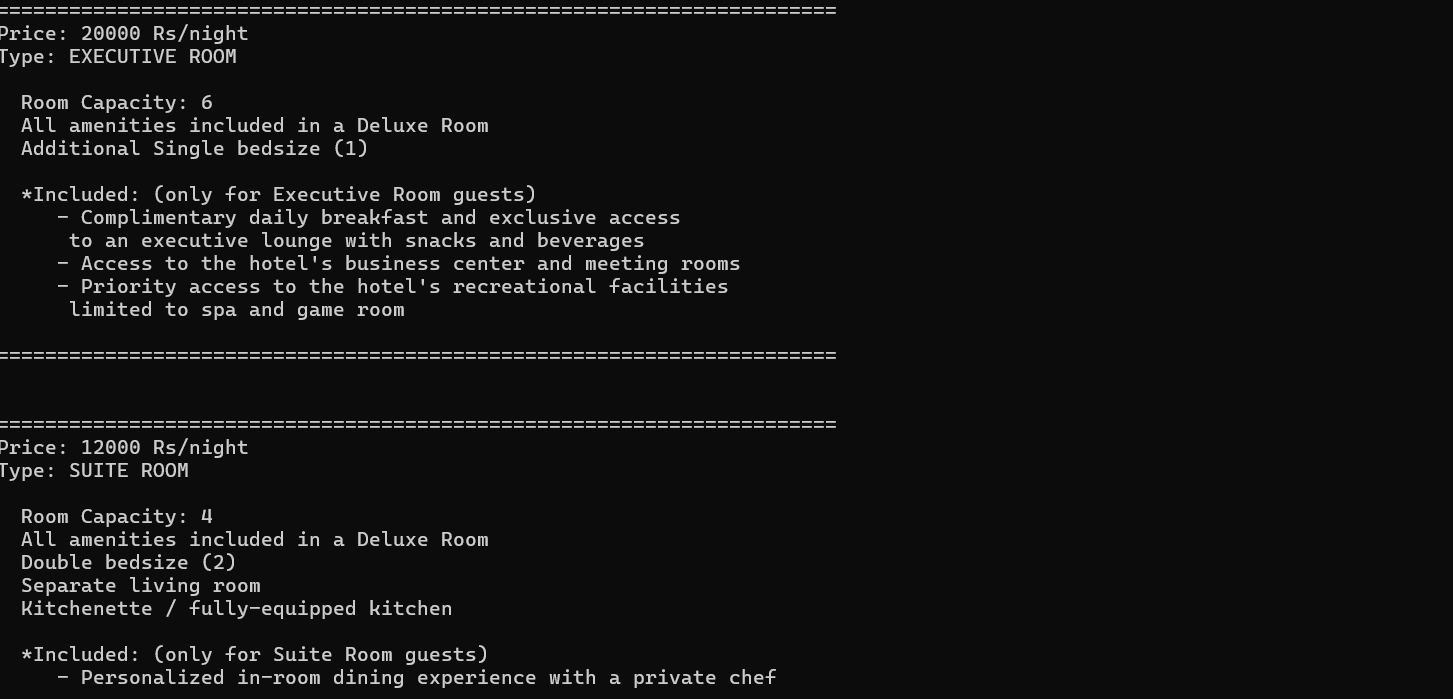
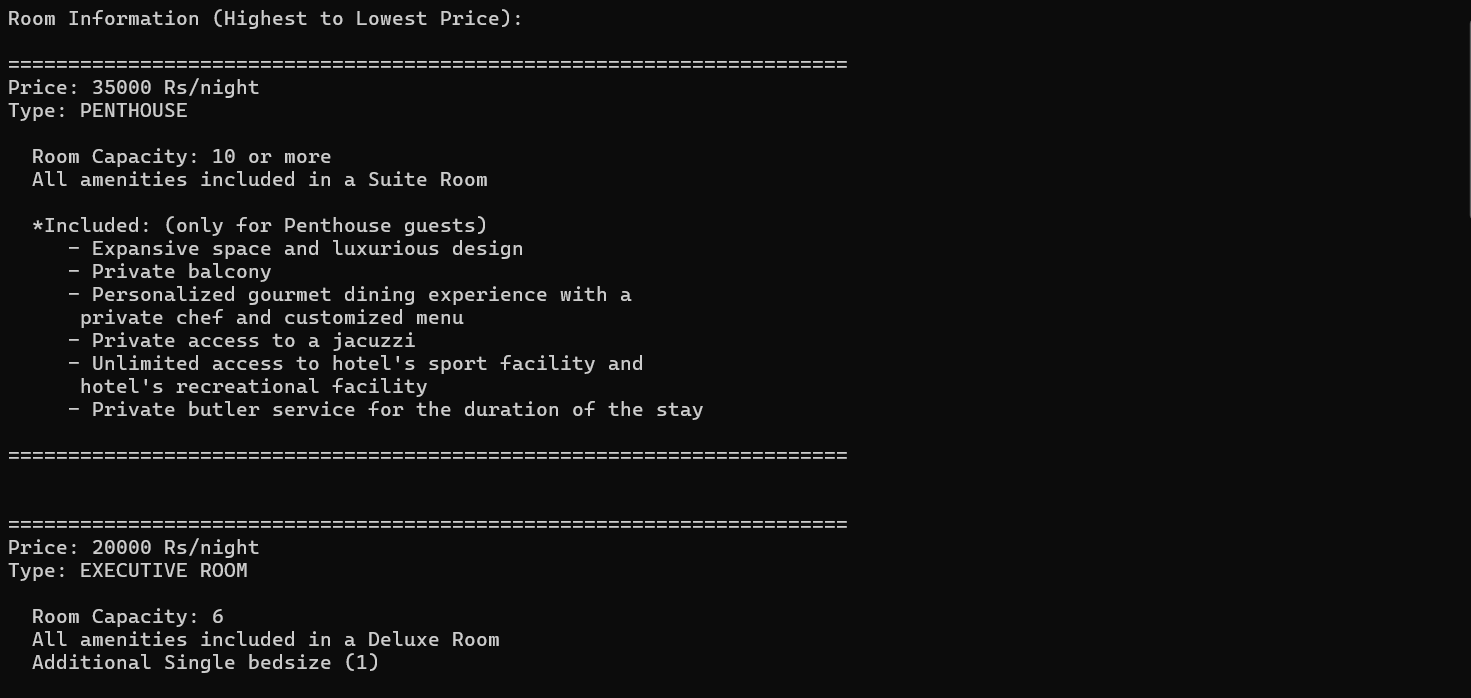
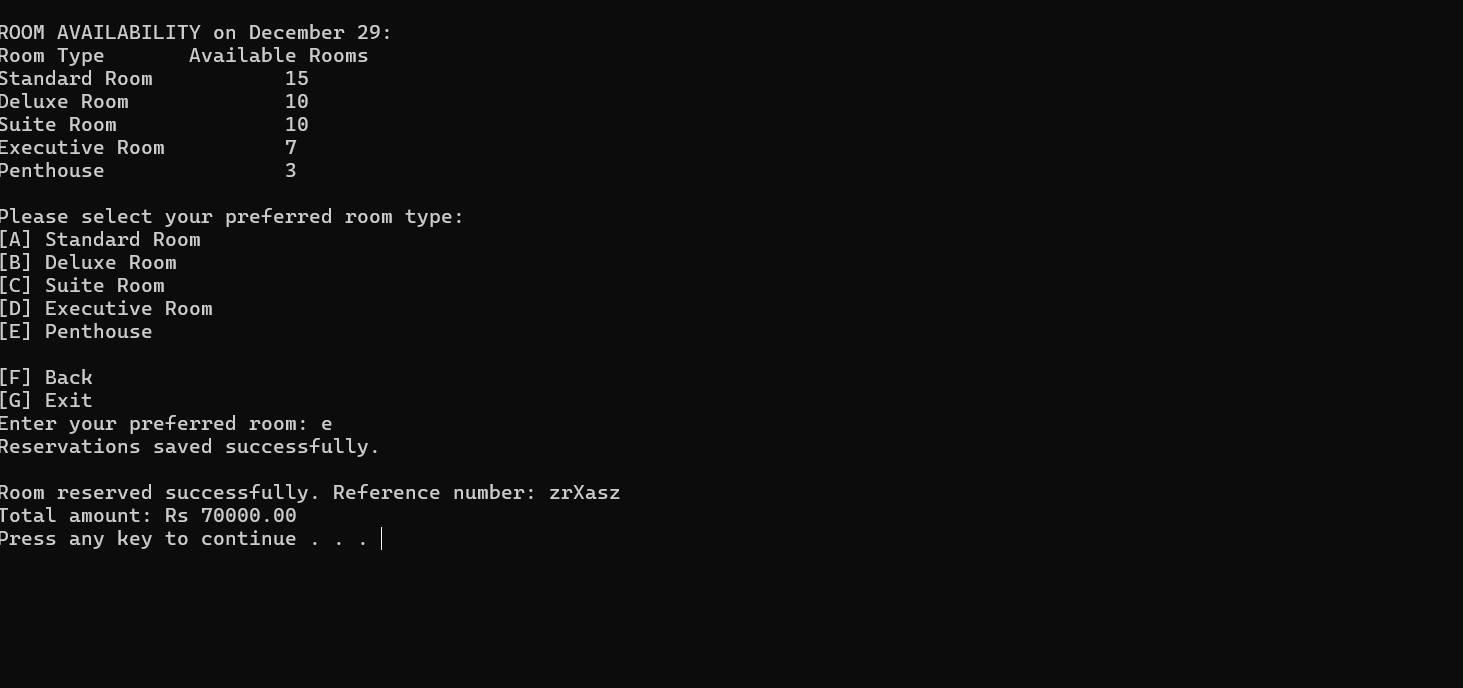
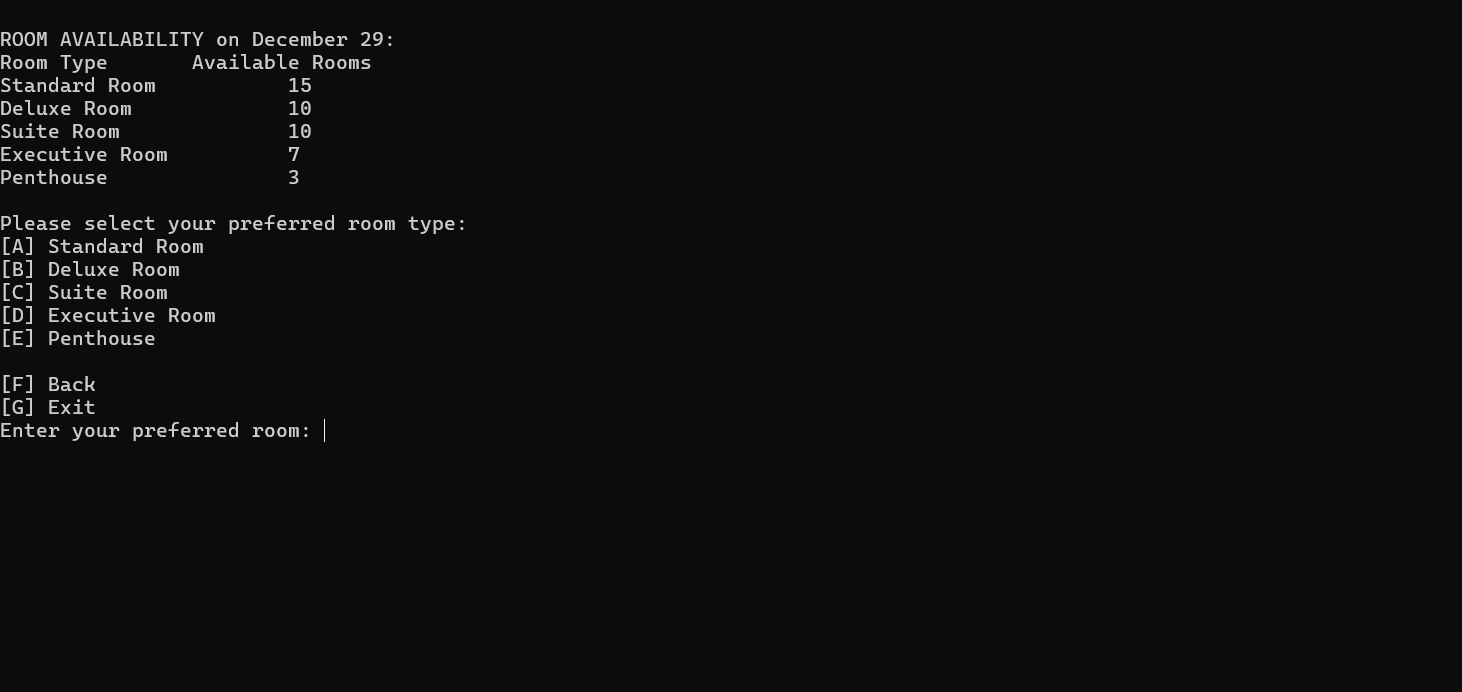
    return availableRooms;

}

* 1. **OUTPUT**



****

****