CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

Hotel Management System

Kavaskar :- 2347230

Josanth Smilan :-2347228

Likitha Yadav G :- 2347234

Submitted to

Dr.B.J.Hubert Shanthan

7th October 2023

TABLE OF CONTENTS

Introduction

Hotel Management System (HMS) is a software solution designed to simplify and improve the operations of hotel organizations. The system serves as a central framework to operate and manage all aspects of hotel operations, including hotel room check-in and check-out, billing, staff management efforts, and general administration. HMS's main goal is to provide guests and hotel staff with a seamless and integrated experience.

The main role of HMS is to manage the room database containing detailed information such as room number, capacity and occupancy rate. It supports the booking process and allows guests to book rooms based on their preferences and availability. The system also collects guest information such as name, check-in date and check-out date, providing a comprehensive view of the hotel's check-in status.

One of the key features of HMS is the ability to improve the billing process. Automatically creates an invoice that includes items such as rooms, additional services, and applicable taxes. This not only reduces manual errors but also speeds up the check-in process and helps improve the guest experience.

Employee management is another important issue that HMS addresses. It allows hotel managers to assign responsibilities, track employee performance and manage working hours efficiently. Additionally, the system helps improve resource allocation by ensuring employees are assigned to specific tasks based on skills.

HMS's integration with databases (often using technologies such as SQLite) allows for easy storage and retrieval of data. This ensures data accuracy, accessibility and security. The system also provides reporting tools that provide information on key performance indicators, helping hotel managers make informed decisions that increase overall profitability.

In short**,** hotel management system is an indispensable tool for modern hotel management. It brings harmony and enjoyment to guests and hotel staff by bringing automation, accuracy and organization to daily hotel operations.

The hotel management system is a complete software solution designed to streamline and automate numerous components of inn operations. This task objectives to beautify the efficiency and business enterprise of responsibilities related to room control, visitor offerings, body of workers management, and facts tracking. underneath is a detailed description of the functionalities and modules within the hotel control system:

Functionalites and Modules

1. Room management Module:

Description:

This module specializes in the effective control of lodge rooms.

Functionalities:

- Room allocation and de-allocation.
- show of room information, along with room number, potential, and occupancy fame.
- Room reservation and availability checking.

2. visitor management Module:

Description:

Manages all guest-related operations and keeps guest records.

Functionalities:

- visitor check-in and take a look at-out methods.
- storage of guest info, along with names, check-in and check-out dates.
- search capability for short retrieval of guest facts.

3. Billing and Invoicing Module:

Description:

Streamlines billing strategies and generates invoices for guests.

Functionalities:

- automatic calculation of prices based on room quotes, extra services, and relevant taxes.
- bill era and printing.
- coping with of payment transactions.

4**. personnel control Module:**

Description:

Manages hotel workforce, their roles, and schedules.

Functionalities:

- body of workers position project and control.
- time table and undertaking allocation for various roles.
- performance tracking and reporting.

5**. Reporting Module:**

Description:

presents insights via reviews on diverse components of lodge operations.

Functionalities:

- era of occupancy reports.
- monetary reviews for sales evaluation.
- team of workers performance reports.

6**. Database Integration:**

Description:

Integrates with a relational database (e.g., SQLite) for green records storage and retrieval.

Functionalities:

- Storing and retrieving information about rooms, guests, and workforce.
- actual-time records access for selection-making.

7**. person Interface (UI) and consumer enjoy (UX):**

Description:

guarantees a person-pleasant interface for each hotel group of workers and guests.

Functionalities:

- Intuitive design for easy navigation.
- Accessibility to all functionalities through a graphical user interface.

8**. safety Module:**

Description:

guarantees the security of touchy statistics in the system.

Functionalities:

- consumer authentication and authorization.
- Encryption of sensitive facts for statistics safety.

9**. Reservation Module:**

Description:

allows room reservations for visitors.

Functionalities:

- on-line reservation machine.
- affirmation and change of reservations.

10**. Integration with outside structures**:

Description:

enables integration with outside structures for a continuing revel in.

Functionalities:

- Integration with online booking systems.
- Connectivity with accounting systems.

The hotel control machine project includes 50 rooms with varying capacities and capabilities an updated code that consists of workforce control for data access and housekeeping roles. It lets in for dynamic updating of personnel records, and all statistics is saved and retrieved from an SQLite database. This complete gadget pursuits to optimize inn operations, enhance visitor offerings, and offer treasured insights for powerful decision-making.

Source Code

```
#include <stdio.h>

#include <string.h>

#include <sqlite3.h>
```

```
// Define enumeration for room status
enum RoomStatus {
    NOT_OCCUPIED,
    OCCUPIED
};

// Define structures
struct Guest {
    int id;
    char name[50];
    char checkInDate[15];
    char checkOutDate[15];
};

struct Staff {
    char name[50];
};

struct Room {
    int roomNumber;
    int capacity;
    enum RoomStatus status;
    struct Guest guest;
    struct Staff dataEntryStaff;
    struct Staff housekeepingStaff;
};

struct Hotel {
    struct Room rooms[50];
```

```
};

// Function prototypes
void initializeRooms(struct Hotel* hotel);
void displayRooms(struct Hotel* hotel);
void checkIn(struct Hotel* hotel, sqlite3* db);
void checkOut(struct Hotel* hotel, sqlite3* db);
void searchGuest(struct Hotel* hotel);
void updateHousekeepingStaff(struct Hotel* hotel);
void updateDataEntryStaff(struct Hotel* hotel);
void saveData(struct Hotel* hotel, sqlite3* db);
void loadData(struct Hotel* hotel, sqlite3* db);

int main() {
    sqlite3* db;
    if (sqlite3_open("hotel.db", &db) != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        return 1;
    }

    struct Hotel hotel;
    initializeRooms(&hotel);
    loadData(&hotel, db);

    int choice;
    do {
        printf("\nHotel Management System\n");
        printf("1. Display Rooms\n");
        printf("2. Check-In\n");
```

```
printf("3. Check-Out\n");
printf("4. Search Guest\n");
printf("5. Update Housekeeping Staff\n");
printf("6. Update Data Entry Staff\n");
printf("7. Save Data\n");
printf("8. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        displayRooms(&hotel);
        break;
    case 2:
        checkIn(&hotel, db);
        break;
    case 3:
        checkOut(&hotel, db);
        break;
    case 4:
        searchGuest(&hotel);
        break;
    case 5:
        updateHousekeepingStaff(&hotel);
        break;
    case 6:
        updateDataEntryStaff(&hotel);
        break;
    case 7:
```

```
                saveData(&hotel, db);
                break;
            case 8:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 8);

    sqlite3_close(db);
    return 0;
}

void initializeRooms(struct Hotel* hotel) {
    for (int i = 0; i < 50; i++) {
        hotel->rooms[i].roomNumber = i + 1;
        hotel->rooms[i].capacity = (i % 5) + 1; // Varying capacities (1 to 5)
        hotel->rooms[i].status = NOT_OCCUPIED; // Not occupied

        // Set default staff names
        sprintf(hotel->rooms[i].dataEntryStaff.name, "DataEntryStaff%d", i + 1);
        sprintf(hotel->rooms[i].housekeepingStaff.name, "HousekeepingStaff%d", i + 1);
    }
}

void displayRooms(struct Hotel* hotel) {
    printf("\nRoom Number\tCapacity\tOccupied\tData Entry Staff\tHousekeeping Staff\n");
    for (int i = 0; i < 50; i++) {
```

```
        printf("%d\t\t%d\t\t%s\t\t%s\t\t%s\n", hotel->rooms[i].roomNumber, hotel->rooms[i].capacity,
               hotel->rooms[i].status == OCCUPIED ? "Yes" : "No",
               hotel->rooms[i].dataEntryStaff.name,hotel->rooms[i].housekeepingStaff.name);
    }
}

void checkIn(struct Hotel* hotel, sqlite3* db) {
    int roomNumber;
    printf("Enter room number for check-in: ");
    scanf("%d", &roomNumber);

    if (roomNumber < 1 || roomNumber > 50) {
        printf("Invalid room number.\n");
        return;
    }

    if (hotel->rooms[roomNumber - 1].status == OCCUPIED) {
        printf("Room is already occupied.\n");
        return;
    }

    printf("Enter guest name: ");
    scanf("%s", hotel->rooms[roomNumber - 1].guest.name);
    printf("Enter check-in date: ");
    scanf("%s", hotel->rooms[roomNumber - 1].guest.checkInDate);
    printf("Enter check-out date: ");
    scanf("%s", hotel->rooms[roomNumber - 1].guest.checkOutDate);
```

```
    hotel->rooms[roomNumber - 1].status = OCCUPIED;
    printf("Check-in successful.\n");
}

void checkOut(struct Hotel* hotel, sqlite3* db) {
    int roomNumber;
    printf("Enter room number for check-out: ");
    scanf("%d", &roomNumber);

    if (roomNumber < 1 || roomNumber > 50) {
        printf("Invalid room number.\n");
        return;
    }

    if (hotel->rooms[roomNumber - 1].status == NOT_OCCUPIED) {
        printf("Room is not occupied.\n");
        return;
    }

    hotel->rooms[roomNumber - 1].status = NOT_OCCUPIED;
    printf("Check-out successful.\n");
}

void searchGuest(struct Hotel* hotel) {
    char guestName[50];
    printf("Enter guest name to search: ");
    scanf("%s", guestName);

    int found = 0;
```

```
    for (int i = 0; i < 50; i++) {
        if (hotel->rooms[i].status == OCCUPIED && strcmp(guestName, hotel->rooms[i].guest.name) == 0) {
            printf("Guest found in room %d\n", hotel->rooms[i].roomNumber);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Guest not found.\n");
    }
}

void updateHousekeepingStaff(struct Hotel* hotel) {
    int roomNumber;
    printf("Enter room number to update housekeeping staff: ");
    scanf("%d", &roomNumber);

    if (roomNumber < 1 || roomNumber > 50) {
        printf("Invalid room number.\n");
        return;
    }

    printf("Enter new housekeeping staff name: ");
    scanf("%s", hotel->rooms[roomNumber - 1].housekeepingStaff.name);
    printf("Housekeeping staff updated for room %d.\n", roomNumber);
}
```

```
void updateDataEntryStaff(struct Hotel* hotel) {
    int roomNumber;
    printf("Enter room number to update data entry staff: ");
    scanf("%d", &roomNumber);

    if (roomNumber < 1 || roomNumber > 50) {
        printf("Invalid room number.\n");
        return;
    }

    printf("Enter new data entry staff name: ");
    scanf("%s", hotel->rooms[roomNumber - 1].dataEntryStaff.name);
    printf("Data entry staff updated for room %d.\n", roomNumber);
}

void saveData(struct Hotel* hotel, sqlite3* db) {
    char* errMsg;
    if (sqlite3_exec(db, "CREATE TABLE IF NOT EXISTS rooms (roomNumber INT, capacity INT, status INT, name TEXT, checkInDate TEXT, checkOutDate TEXT, dataEntryStaff TEXT, housekeepingStaff TEXT);", NULL, NULL, &errMsg) != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", errMsg);
        sqlite3_free(errMsg);
        return;
    }

    sqlite3_exec(db, "BEGIN TRANSACTION", 0, 0, 0);

    for (int i = 0; i < 50; i++) {
        char query[400];
```

```
        sprintf(query, "INSERT INTO rooms (roomNumber, capacity, status, name, checkInDate, checkOutDate, dataEntryStaff, housekeepingStaff) VALUES (%d, %d, %d, '%s', '%s', '%s', '%s', '%s');",
                hotel->rooms[i].roomNumber,hotel->rooms[i].capacity, hotel->rooms[i].status,
                hotel->rooms[i].guest.name,hotel->rooms[i].guest.checkInDate,hotel->rooms[i].guest.checkOutDate,
                hotel->rooms[i].dataEntryStaff.name,hotel-->rooms[i].housekeepingStaff.name);

        if (sqlite3_exec(db, query, NULL, NULL, &errMsg) != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", errMsg);
            sqlite3_free(errMsg);
        }
    }

    sqlite3_exec(db, "COMMIT", 0, 0, 0);
    printf("Data saved successfully.\n");
}

void loadData(struct Hotel* hotel, sqlite3* db) {
    sqlite3_stmt* stmt;
    char query[] = "SELECT roomNumber, capacity, status, name, checkInDate, checkOutDate, dataEntryStaff, housekeepingStaff FROM rooms;";

    if (sqlite3_prepare_v2(db, query, -1, &stmt, 0) != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", sqlite3_errmsg(db));
        return;
    }

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        int roomNumber = sqlite3_column_int(stmt, 0);
        hotel->rooms[roomNumber - 1].capacity = sqlite3_column_int(stmt, 1);
```

```
        hotel->rooms[roomNumber - 1].status = sqlite3_column_int(stmt, 2);
        strcpy(hotel->rooms[roomNumber - 1].guest.name, sqlite3_column_text(stmt, 3));
        strcpy(hotel->rooms[roomNumber-1].guest.checkInDate, sqlite3_column_text(stmt, 4));
        strcpy(hotel->rooms[roomNumber-1].guest.checkOutDate,      sqlite3_column_text(stmt,
5));
        strcpy(hotel->rooms[roomNumber-1].dataEntryStaff.name,     sqlite3_column_text(stmt,
6));
        strcpy(hotel->rooms[roomNumber-1].housekeepingStaff.name,
sqlite3_column_text(stmt, 7));
    }

    sqlite3_finalize(stmt);
    printf("Data loaded successfully.\n");
}
```

ScreenShots

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 2
Enter room number for check-in: 7
Enter guest name: likitha
Enter check-in date: 12
Enter check-out date: 18
Check-in successful.
```

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 3
Enter room number for check-out: 4
Check-out successful.
```

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 4
Enter guest name to search: kavaskar
Guest found in room 1
```

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 5
Enter room number to update housekeeping staff: 4
Enter new housekeeping staff name: aasha
Housekeeping staff updated for room 4.
```

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 6
Enter room number to update data entry staff: 4
Enter new data entry staff name: likitha
Data entry staff updated for room 4.
```

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 7
Data saved successfully.
```

```
Hotel Management System
1. Display Rooms
2. Check-In
3. Check-Out
4. Search Guest
5. Update Housekeeping Staff
6. Update Data Entry Staff
7. Save Data
8. Exit
Enter your choice: 8
Exiting the program.
```

DataBase

1	1	0	likitha	12	17	Datal
2	2	1	likitha	15	20	Datal
3	3	1	likitha	1	10	Datal
4	4	1	likitha	2	12	likith
5	5	1	likitha	3	14	Datal
6	1	1	likitha	4	16	Datal
7	2	1	smilan	1	3	Datal
8	3	1	smilan	5	8	Datal
9	4	1	smilan	9	14	Datal
10	5	1	kavaskar	17	24	Datal
11	1	1	kavaskar	25	29	Datal
12	2	1	kavaskar	26	29	Datal
13	3	1	kavaskar	12	14	Datal
14	4	1	likitha	14	19	Datal
15	5	1	kalpu	12	15	Datal
16	1	1	priya	7	9	Datal
17	2	1	lahari	18	19	Datal

Future Enhancements

The hotel control device defined above serves as a strong foundation for motel operations, but there's always room for development and growth. right here are a few future upgrades that would be considered for similarly improvement:

1.**online booking and Reservation device:**

- put into effect an online portal for guests to make reservations, view room availability, and manage bookings remotely.

2. **cell utility:**

- develop a cell app to offer visitors and team of workers with the ability to control reservations, check-in, and access motel offerings through their smartphones.

3**. Integration with smart devices:**

- incorporate internet of factors (IoT) gadgets for clever room controls, allowing guests to modify lighting, temperature, and other room settings via mobile apps.

4. **AI-Powered Chatbot:**

- integrate an AI-powered chatbot to offer instant responses to guest inquiries, deal with commonplace requests, and assist with data retrieval.

5. **superior Analytics and commercial enterprise Intelligence:**

- beautify reporting capabilities with superior analytics equipment to provide deeper insights into guest preferences, occupancy traits, and sales patterns.

6**. Loyalty program Integration:**

- integrate a loyalty software to praise common visitors, encourage repeat enterprise, and collect valuable consumer data for personalised services.

7. **Multi-region management:**

- extend the machine to manage multiple motel places, allowing centralized manage and reporting for hotel chains.

8. **better security functions:**

- strengthen security features with biometric authentication for team of workers get right of entry to, surveillance camera integration, and at ease price gateways.

9**. automatic take a look at-In and take a look at-Out:**

- discover technologies which include facial recognition or RFID playing cards for streamlined and contactless check-in and take a look at-out processes.

10**. Language support and Localization:**

- provide guide for more than one languages to accommodate global guests, and bear in mind localization features for cultural preferences.

11**. electricity performance Measures:**

- combine energy management systems to optimize strength intake in unoccupied rooms and make contributions to sustainability goals.

persevered innovation and model to rising technologies will make certain that the hotel control device remains aggressive and meets the evolving needs of the hospitality industry.

Conclusion:

In conclusion, the hotel management device provides a strong and complete answer for green motel operations. by incorporating features such as room control, visitor test-in and check-out, and workforce assignments, the gadget presents a streamlined method to hotel administration. The inclusion of a SQLite database guarantees records endurance and allows for seamless information retrieval.

The device's consumer-friendly interface makes it handy for each guests and lodge personnel, improving the general consumer enjoy. The implementation of facts access and housekeeping workforce, together with their next updates, contributes to effective staff control. furthermore, the mixing of SQLite allows information storage, retrieval, and management.

however, there's continually room for improvement and enlargement. destiny enhancements should encompass on line reserving abilities, cellular applications, and integration with smart devices to elevate the guest enjoy. moreover, advanced analytics and security capabilities may want to similarly optimize resort operations and visitor safety.

In essence, the lodge control system serves as a basis for digitalizing and automating key resort strategies. Its modern-day skills cope with fundamental lodge management wishes, laying the groundwork for potential advancements in the ever-evolving hospitality enterprise.