**Computer PROGRAMMING**

**--Hospital Management System --**

**Project Report**

Table of Content:

[Instructions 3](#_Toc155457424)

[Introduction: 4](#_Toc155457425)

[ Background 4](#_Toc155457426)

[ Scope 4](#_Toc155457427)

[ Objectives 4](#_Toc155457428)

[Methodology: 4](#_Toc155457429)

[Implementation: 4](#_Toc155457430)

[ Inputs 4](#_Toc155457431)

[ Outputs 4](#_Toc155457432)

[ Code Snippets 4](#_Toc155457433)

[Conclusion: 4](#_Toc155457434)

[ Evaluation 4](#_Toc155457435)

[ Screenshots 4](#_Toc155457436)

**INTRODUCTION:**

In the fast-paced world of healthcare, effective management of hospital resources is paramount to ensure the delivery of quality patient care. The Hospital Management System (HMS) serves as a comprehensive solution designed to streamline and optimize the daily operations of a healthcare facility. Developed using the C++ programming language, this project aims to enhance the efficiency, accuracy, and accessibility of managing various aspects of hospital administration. The inclusion of robust security measures guarantees the confidentiality and integrity of sensitive patient information, aligning with data protection standards.

The strict access control measures further ensure that doctors can access data only for their designated patients, enhancing confidentiality and compliance with privacy regulations. By incorporating features such as patient feedback solicitation, room occupancy management, and dynamic doctor information tracking, our system is poised to elevate the standards of healthcare management. This project not only embraces technological advancements but also prioritizes patient confidentiality, privacy, and the seamless flow of medical information.

**Background:**

The healthcare industry plays a critical role in society, providing essential services to individuals in need. However, the conventional methods of managing patient information, doctor appointments, and overall healthcare facility operations have often proven to be inefficient, time-consuming, and susceptible to errors. In an era where technology can significantly enhance the efficiency and efficacy of healthcare services, there is a pressing need for a comprehensive Healthcare Management System.

**Problem Statement:**

The existing healthcare management practices are burdened with manual record-keeping, outdated appointment systems, and a lack of systematic approaches to gather patient feedback and manage financial aspects. These inefficiencies not only hinder the daily operations of healthcare facilities but also compromise the quality of patient care. In light of these challenges, our project aims to address these issues by developing a robust Healthcare Management System that automates key processes, streamlines data management, and prioritizes patient-centric care.

**SCOPE:**

Our HealthCare Management System is a comprehensive solution tailored to meet the evolving needs of major healthcare departments, encompassing Nutrition, Ophthalmology, OPD (Outpatient Department), Gynecology, Dermatology, Cardiology, Nephrology, Infectious Diseases/Viral, Otorhinolaryngology (E.N.T), and Physiotherapy. Our HealthCare Management System is designed to revolutionize the management of major hospital departments, ensuring efficient, secure, and patient-centric healthcare services. This system aims to uphold the highest standards of medical practice while embracing technological advancements for the betterment of patient care. The strict access control measures further ensure that doctors can access data only for their designated patients, enhancing confidentiality and compliance with privacy regulations. By incorporating features such as patient feedback solicitation, room occupancy management, and dynamic doctor information tracking, our system is poised to elevate the standards of healthcare management. This project not only embraces technological advancements but also prioritizes patient confidentiality, privacy, and the seamless flow of medical information.

**OBJECTIVES:**

The HealthCare Management System is imperative for its pivotal role in revolutionizing the efficiency and efficacy of healthcare operations. In the context of our project, this system serves as an indispensable tool for storing, accessing, protecting, updating, deleting, and viewing patient data with unparalleled precision. Saving time, money (which would be otherwise used to manage these records, and buying papers and documents to physically record the information), and energy of the staff (which would have been appointed to manage these records) allows a shift in focus. Instead of spending so much time on paperwork, healthcare workers can now concentrate on improving patient care and making healthcare facilities better. This positive impact goes beyond healthcare, creating opportunities for initiatives that care for the environment and create spaces that are eco-friendly.

**Update Patient's Information:**

* **Objective:** Ensure the accuracy and currency of patient records by implementing a system that allows for the regular update of patient information.
* **Implementation Steps:**
  + Develop a secure database to store patient information.
  + Provide user interfaces for healthcare staff to input and update patient details.
  + Ensure compliance with privacy and security regulations.

**Allowing Patients to Select the Doctor for Consultation and Booking Their Appointment:**

* **Objective:** Facilitate a user-friendly system that enables patients to choose their preferred doctor and book appointments efficiently.
* **Implementation Steps:**
  + Create a user interface for patients to browse through a list of available doctors.
  + Send confirmation notifications to patients upon successful booking.

**Managing Doctors' Information:**

* **Objective:** Develop a robust system for the management of doctors' information, including adding, updating, and deleting data as needed.
* **Implementation Steps:**
  + Design a database to store comprehensive information about each doctor, including name, specialization, and department.
  + Create interfaces for administrators or authorized personnel to add, update, or remove doctor information.
  + Implement version control to track changes made to doctor records for accountability.

**Taking Feedback from the Patient:**

* **Objective:** Establish a feedback mechanism to gather patient opinions and experiences for continuous improvement.
* **Implementation Steps:**
  + Design feedback forms or surveys for patients to complete after consultations.
  + Use feedback to identify areas of improvement and enhance the quality of healthcare services.
* **Room Occupancy Management:**
* **Objective:** Efficiently manage the occupancy of rooms within the healthcare facility for various purposes.
* **Implementation Steps:**
  + Develop a system to monitor real-time room occupancy.
  + Integrate a scheduling mechanism to allocate rooms for consultations, treatments, and other activities.

**To Keep Track of Finance Generated by the Consultation of Each Department:**

* **Objective:** Establish a financial tracking system to monitor revenue generated from consultations in each department.
* **Implementation Steps:**
  + Integrate the systems to record financial transactions related to consultations.
  + Categorize revenue by department to track performance.
  + Generate financial reports and analytics to aid in decision-making.
  + Ensure compliance with financial regulations and reporting standards.

**INPUTS AND OUTPUTS:**

1. **Update Patient's Information:**
   * Inputs:
     + Patient details (name, ID, etc.)
     + Updated information
   * Outputs:
     + Updated patient records in the secure database
     + Confirmation/notification of successful update
2. **Allowing Patients to Select the Doctor for Consultation and Booking Their Appointment:**
   * Inputs:
   * Patient preferences (selected doctor)
   * Outputs:
     + Confirmation/notification of successful appointment booking
3. **Managing Doctors' Information:**
   * Inputs:
     + Doctor details (name, specialization, department, etc.)
     + Requests for adding, updating, or deleting doctor information
   * Outputs:
     + Updated doctor records in the database
     + Confirmation/notification of successful update or deletion
4. **Taking Feedback from the Patient:**
   * Inputs:
     + Patient feedback forms or surveys
   * Outputs:
     + Aggregated feedback data for analysis
     + Identified areas of improvement
5. **Room Occupancy Management:**
   * Inputs:
     + Real-time occupancy data
     + Scheduling requests for room allocation
   * Outputs:
     + Updated room occupancy status
     + Room booked for patient
6. **To Keep Track of Finance Generated by the Consultation of Each Department:**
   * Inputs:
     + Financial transactions related to consultations
     + Categorization of revenue by department
   * Outputs:
     + Financial reports and analytics
     + Compliance reports with financial regulations and standards

**Methodology and Implementation:**

1. **File Handling (fstream):**
   * **Purpose:** To store and retrieve patient and doctor information persistently.
   * **Usage:**
     + Utilize file handling (fstream) to create, read, write, and update records in external files.
     + Implement functions for reading and writing data to files, ensuring data persistence between program runs.
     + Leverage the fstream library for seamless file operations.
2. **Arrays:**
   * **Purpose:** Efficient storage and manipulation of patient and doctor data.
   * **Usage:**
     + Create arrays to store and manage data for doctors and patients, providing a structured format.
     + Utilize arrays for searching, updating, and deleting records based on user input.
     + Implement loops for iterating through arrays, making it easier to perform repetitive tasks and manage data systematically.
3. **For Loop:**
   * **Purpose:** Iterative processing for data entry, searching, updating, and deletion.
   * **Usage:**
     + Employ for loops for efficient iteration through arrays, enabling streamlined data entry for doctors and patients.
     + Utilize for loops for searching specific records based on user input.
     + Enable easy updates or deletions by combining for loops with conditional statements.
4. **While Loop:**
   * **Purpose:** Continuously execute specific functionalities until the user decides to exit.
   * **Usage:**
     + Implement while loops to create a user-friendly interface that allows continuous interaction until the user chooses to exit.
     + Utilize while loops for taking user input, providing a dynamic and responsive program flow.
5. **String Library:**
   * **Purpose:** Efficient handling of textual data such as names, specializations, and feedback.
   * **Usage:**
     + Utilize the string library for handling and manipulating strings, ensuring accurate representation of textual data.
     + Employ string variables for storing and processing names, feedback, and other textual information.

**ALGORITHM :**

* + - * **Initialization:**

Define constants MAX\_DOCTORS and MAX\_PATIENTS.

Define structures Doctor and Patient with relevant attributes.

Declare arrays doc for doctors and pat for patients.

Declare global variables doctorCount and patientCount.

* + - * **File Handling Functions:**

Implement readData function to read data from files (doctors.txt and patients.txt) into arrays.

Implement writeDoc and writepatient functions to write doctor and patient data to files, respectively.

* + - * **Administrator Functions:**
* Implement admin function for administrator tasks.
* Inside admin:
* Display welcome message.
* Use a do-while loop to present a menu with options:

1. Add doctor
2. Remove doctor
3. Display finance generated by each doctor
4. Display all doctors with rating
5. Exit

Call doctor\_info based on the user's choice.

* + - * **Receptionist Functions:**
* Implement receptionist function for receptionist tasks.
* Inside receptionist:
* Display welcome message.
* Use a do-while loop to present a menu with options:

1. Add patient
2. Enter health concern
3. Display list of all doctors
4. Display patient history
5. Exit

Call patient\_info based on the user's choice.

* + - * **Doctor Functions:**
* Implement doctorFunc function for doctor tasks.
* Inside doctorFunc:
* Display welcome message.
* Use a do-while loop to present a menu with options:
* View patient history
* Update patient history
* Exit

Call doctor\_ based on the user's choice.

* **Patient Functions:**
* Implement patientFunc function for patient tasks.
* Inside patientFunc:
* Display welcome message.
* Use a do-while loop to present a menu with options:

1. View medical history
2. Enter health concern
3. Display list of all doctors
4. Book your Appointment
5. Add rating of visited doctor
6. Exit

Call patient\_ based on the user's choice.

* **Doctor Information Functions:**
* Implement doctor\_info function for handling doctor-related information.
* Inside doctor\_info:
* Implement options to add, remove, display finance, or display doctors with rating.
* Call writeDoc after adding or removing a doctor.
* **Patient Information Functions:**
* Implement patient\_info function for handling patient-related information.
* Inside patient\_info:
* Implement options to add patient, enter health concern, display doctors, or display patient history.
* Call writepatient after adding a patient.
* **Doctor-specific Functions:**

Implement doctor\_ function for doctor-specific tasks (view/update patient history).

* **Patient-specific Functions:**

Implement patient\_ function for patient-specific tasks (view medical history, enter health concern, display doctors, book appointment, add rating).

* **Main Function:**
* Implement main function.
* Call readData to load existing data.
* Use a do-while loop to present the main menu with options:

1. Admin
2. Receptionist
3. Doctor
4. Patient
5. Exit

Call the respective functions based on the user's choice.

This algorithm provides a high-level overview of the code structure and functionality. It can serve as a guide for understanding the flow of the program.

**Flowcharts:**



**Code Snippets:**

#include <iostream>

#include <windows.h>

#include <string>

#include <fstream>

using namespace std;

const int MAX\_DOCTORS = 50;

const int MAX\_PATIENTS = 50;

struct Doctor

{

string name, department, speciality;

int fees = 0, rating = 0, id = 0, counter = 0, rate = 0;

int sum;

};

struct Patient

{

string name, concern, medical;

int id = 0, visit = 0;

char appointment = '\0';

};

Doctor doc[MAX\_DOCTORS];

int doctorCount = 0;

Patient pat[MAX\_PATIENTS];

int patientCount = 0;

void admin();

void receptionist();

void doctorFunc();

void patientFunc();

void doctor\_info(int choice);

void patient\_info(int choice);

void doctor\_(int choice);

void patient\_(int choice);

void readData();

//void writeData();

void readData()

{

ifstream doctorFile("doctors.txt");

ifstream patientFile("patients.txt");

doctorFile >> doctorCount;

for (int i = 0; i < doctorCount; i++)

{

doctorFile >> doc[i].id >> doc[i].name >> doc[i].department >> doc[i].speciality >> doc[i].fees >> doc[i].rating >> doc[i].counter >> doc[i].sum;

}

doctorFile.close();

patientFile >> patientCount;

for (int i = 0; i < patientCount; i++)

{

patientFile >> pat[i].id >> pat[i].name >> pat[i].concern >> pat[i].medical >> pat[i].visit >> pat[i].appointment;

}

patientFile.close();

}

void writeDoc()

{

ofstream doctorFile("doctors.txt",ios::app);

doctorFile << doctorCount << endl;

for (int i = 0; i < doctorCount; i++)

{

// Writing doctor information to the file

doctorFile << doc[i].id << " " << doc[i].name << " " << doc[i].department << " "

<< doc[i].speciality << " " << doc[i].fees << " "

<< doc[i].rating << " " << doc[i].counter << " " << doc[i].sum << endl;

}

doctorFile.close();

}

void writepatient() {

ofstream patientFile("patients.txt",ios::app);

patientFile << patientCount << endl;

for (int i = 0; i < patientCount; i++)

{

// Writing patient information to the file

patientFile << pat[i].id << " " << pat[i].name << " " << pat[i].concern << " "

<< pat[i].medical << " " << pat[i].visit << " " << pat[i].appointment << endl;

}

patientFile.close();

}

void admin()

{

cout << "\n\n\n\n\n\t\t\t Loading... !\n";

char x = '\*';

for (int i = 0; i <= 27; i++)

{

Sleep(100);

cout << x;

}

system("cls");

cout << "\n\n\t\t\tWelcome ADMIN!!" << endl;

int choice;

do

{

cout << "ADMIN" << endl;

cout << "1.Add doctor\n";

cout << "2.Remove doctor\n";

cout << "3.Display finance generated by each doctor\n";

cout << "4.Display all doctors with rating\n";

cout << "5.Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice != 5)

{

doctor\_info(choice);

}

} while (choice != 5);

}

void receptionist()

{

cout << "\n\n\n\n\n\t\t\t Loading... !\n";

char x = '\*';

for (int i = 0; i <= 27; i++)

{

Sleep(100);

cout << x;

}

system("cls");

cout << "\n\n\t\t\tWelcome RECEPTIONIST!!" << endl;

int choice;

do

{

cout << "RECEPTIONIST" << endl;

cout << "1.Add patient\n";

cout << "2.Enter your health concern\n";

cout << "3.Display list of all doctors\n";

cout << "4.Display patient history\n";

cout << "5.Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice != 5)

{

patient\_info(choice);

}

} while (choice != 5);

}

void doctorFunc()

{

cout << "\n\n\n\n\n\t\t\t Loading.. !\n";

char x = '\*';

for (int i = 0; i <= 27; i++)

{

Sleep(100);

cout << x;

}

system("cls");

cout << "\n\n\t\t\tWelcome DOCTOR!!" << endl;

int choice;

do

{

cout << "DOCTOR" << endl;

cout << "1.View patient history\n";

cout << "2.Update patient history\n";

cout << "3.Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice != 3)

{

doctor\_(choice);

}

} while (choice != 3);

}

void patientFunc()

{

cout << "\n\n\n\n\n\t\t\t Loading.. !\n";

char x = '\*';

for (int i = 0; i <= 27; i++)

{

Sleep(100);

cout << x;

}

system("cls");

cout << "\n\n\t\t\tWelcome PATIENT!!" << endl;

int choice;

do {

cout << "PATIENT" << endl;

cout << "1.View medical history\n";

cout << "2.Enter health concern\n";

cout << "3.Display list of all doctors\n";

cout << "4.Book your Appointment\n";

cout << "5.Add rating of visited doctor\n";

cout << "6.Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice != 6)

{

patient\_(choice);

}

} while (choice != 6);

}

void doctor\_info(int choice)

{

if (choice == 1)

{

cout << endl;

for (int i = doctorCount; i < 50; i++)

{

cout << endl;

cin.ignore();

cout << "Enter doctor id: ";

cin >> doc[i].id;

cin.ignore();

cout << "Enter doctor name: ";

getline(cin, doc[i].name);

cout << "Enter doctor department: ";

getline(cin, doc[i].department);

cout << "Enter doctor speciality: ";

getline(cin, doc[i].speciality);

cout << "Enter doctor fees: ";

cin >> doc[i].fees;

doc[i].rating = 0;

doctorCount++;

writeDoc();

cout << "Do you want to add another doctor? (1 for yes, 0 for no): ";

int addAnother;

cin >> addAnother;

if (addAnother != 1)

{

break;

}

cout << endl;

}

if (doctorCount == 50)

{

cout << "Maximum number of doctors added. Cannot add more.\n";

}

else

{

cout << "Added successfully\n";

}

cout << endl;

}

else if (choice == 3)

{

cout << endl;

if (doctorCount == 0)

{

cout << "No doctors available." << endl;

}

else

{

for (int i = 0; i < doctorCount; i++)

{

cout << "Doctor ID: " << doc[i].id << endl;

cout << "Doctor Name: " << doc[i].name << endl;

cout << "Doctor Department: " << doc[i].department << endl;

cout << "Doctor finance: " << doc[i].sum << endl;

cout << endl;

cout << endl;

}

}

cout << endl;

}

else if (choice == 4)

{

cout << endl;

if (doctorCount == 0)

{

cout << "No doctors available." << endl;

}

else

{

for (int i = 0; i < doctorCount; i++)

{

cout << "Doctor ID: " << doc[i].id << endl;

cout << "Doctor Name: " << doc[i].name << endl;

cout << "Doctor Department: " << doc[i].department << endl;

cout << "Doctor Speciality: " << doc[i].speciality << endl;

cout << "Doctor Fees: " << doc[i].fees << endl;

cout << "Doctor Rating: " << doc[i].rating << endl;

cout << "Doctor finance: " << doc[i].sum << endl;

cout << "NO OF PATIENT VISITED: " << doc[i].counter << endl;

cout << endl;

}

}

cout << endl;

}

else if (choice == 2)

{

int a;

cout << "Enter id to delete: ";

cin >> a;

bool doctorFound = false;

for (int i = 0; i < doctorCount; i++)

{

if (a == doc[i].id)

{

for (int j = i; j < doctorCount - 1; j++)

{

doc[j] = doc[j + 1];

}

doctorCount--;

doctorFound = true;

break;

}

}

if (doctorFound)

{

cout << "Deleted successfully\n";

}

else

{

cout << "Doctor with ID " << a << " not found\n";

}

cout << endl;

}

else

{

cout << "invalid choice\n";

cout << endl;

}

}

void patient\_info(int choice)

{

if (choice == 1)

{

cout << endl;

for (int i = patientCount; i < MAX\_PATIENTS; i++)

{

cout << "Enter patient id: ";

cin >> pat[i].id;

cin.ignore();

cout << "Enter patient name: ";

getline(cin, pat[i].name);

writepatient();

cout << "Do you want to add another patient? (1 for yes, 0 for no): ";

int addAnother;

cin >> addAnother;

if (addAnother != 1)

{

break;

}

patientCount++;

}

if (patientCount == MAX\_PATIENTS)

{

cout << "Maximum number of patients added. Cannot add more.\n";

}

else

{

cout << "Added successfully\n";

}

cout << endl;

}

else if (choice == 2)

{

cout << endl;

int patientId;

cout << "Enter patient id: ";

cin >> patientId;

bool

patientFound = false;

for (int i = 0; i < 5; i++)

{

if (patientId == pat[i].id)

{

cout << "Enter patient concern: ";

cin >> pat[i].concern;

pat[i].visit++;

bool doctorFound = false;

for (int j = 0; j < 50; j++)

{

if (pat[i].concern == doc[j].speciality || pat[i].concern == doc[j].department)

{

cout << endl;

cout << "DOCTOR FOR YOUR CONCERN\*\n";

cout << "Doctor ID: " << doc[j].id << endl;

cout << "Doctor Name: " << doc[j].name << endl;

cout << "Doctor Department: " << doc[j].department << endl;

cout << "Doctor Speciality: " << doc[j].speciality << endl;

cout << "Doctor Fees: " << doc[j].fees << endl;

doctorFound = true;

cout << "Do you want to book an appointment with " << doc[j].name << "? (y/n): ";

cin >> pat[i].appointment;

if (pat[i].appointment == 'Y' || pat[i].appointment == 'y')

{

doc[j].sum += doc[j].fees;

cout << "Booked\n";

}

else

{

break;

}

break;

}

}

if (!doctorFound)

{

cout << "No matching doctor found" << endl;

}

patientFound = true;

break;

}

}

if (!patientFound)

{

cout << "Patient with ID " << patientId << " not found" << endl;

}

cout << endl;

}

else if (choice == 3)

{

cout << endl;

for (int i = 0; i < doctorCount; i++)

{

cout << "Doctor ID: " << doc[i].id << endl;

cout << "Doctor Name: " << doc[i].name << endl;

cout << "Doctor Department: " << doc[i].department << endl;

cout << "Doctor Speciality: " << doc[i].speciality << endl;

cout << "Doctor Fees: " << doc[i].fees << endl;

cout << endl;

}

cout << endl;

}

else if (choice == 4)

{

cout << endl;

int patientId;

cout << "Enter patient id: ";

cin >> patientId;

bool patientFound = false;

for (int i = 0; i < 50; i++)

{

if (patientId == pat[i].id)

{

cout << "PATIENT ID: " << pat[i].id << endl;

cout << "PATIENT NAME: " << pat[i].name << endl;

cout << "NO OF VISITS: " << pat[i].visit << endl;

cout << "MEDICAL HISTORY: " << pat[i].medical << endl;

patientFound = true;

break;

}

}

if (!patientFound)

{

cout << "Patient not found\n";

}

cout << endl;

}

else

{

cout << endl;

cout << "inavlid choice\n";

cout << endl;

}

}

void doctor\_(int choice)

{

int a;

cout << "Enter patient's id: ";

cin >> a;

if (choice == 1)

{

bool patientFound = false;

for (int i = 0; i < 50; i++)

{

if (a == pat[i].id)

{

string c = pat[i].concern;

if (pat[i].appointment == 'y' || pat[i].appointment == 'Y')

{

string c = pat[i].concern;

cout << "Patient name: " << pat[i].name << endl;

cout << "Patient's concern: " << pat[i].concern << endl;

cout << "Medical history: " << pat[i].medical << endl;

patientFound = true;

break;

}

}

}

if (!patientFound)

{

cout << " Appointment not booked\n";

}

cout << endl;

}

else if (choice == 2)

{

cout << endl;

cout << "Enter patient's id: ";

cin >> a;

bool appointmentFound = false;

for (int i = 0; i < 50; i++)

{

if (pat[i].id == a && (pat[i].appointment == 'y' || pat[i].appointment == 'Y'))

{

cout << "Enter prescription or test if needed: ";

cin.ignore();

getline(cin, pat[i].medical);

appointmentFound = true;

break;

}

}

if (!appointmentFound)

{

cout << "Appointment not booked or patient not found\n";

}

cout << endl;

}

else

{

cout << endl;

cout << "invalid choice\n";

cout << endl;

}

}

void patient\_(int choice)

{

int a;

if (choice == 1)

{

cout << endl;

for (int i = 0; i < 50; i++)

{

cout << "Enter your id: ";

cin >> a;

if (pat[i].id == a)

{

cout << "ID: " << pat[i].id << endl;

cout << "NAME: " << pat[i].name << endl;

cout << "MEDICAL HISTORY: " << pat[i].medical << endl;

cout << "NO OF VISITS: " << pat[i].visit << endl;

}

else

{

cout << "patient not found\n";

}

break;

}

cout << endl;

}

else if (choice == 2)

{

cout << endl;

string c;

bool found = false;

cin.ignore();

cout << "Enter your concern: ";

getline(cin, c);

for (int i = 0; i < 50; i++)

{

if (c == doc[i].speciality || c == doc[i].department)

{

found = true;

cout << "Doctor ID: " << doc[i].id << endl;

cout << "Doctor Name: " << doc[i].name << endl;

cout << "Doctor Department: " << doc[i].department << endl;

cout << "Doctor Speciality: " << doc[i].speciality << endl;

cout << "Doctor Fees: " << doc[i].fees << endl;

cout << "Want to book an appointment? (y/n): ";

cin >> pat[i].appointment;

if (pat[i].appointment == 'y' || pat[i].appointment == 'Y')

{

cout << "Book your appointment by visiting the receptionist. Thank you." << endl;

}

else

{

cout << "Thank you." << endl;

}

break;

}

}

if (!found)

{

cout << "No doc to display for your concern" << endl;

}

cout << endl;

}

else if (choice == 3)

{

cout << endl;

if (doctorCount != 0)

{

for (int i = 0; i < doctorCount; i++)

{

cout << "Doctor ID: " << doc[i].id << endl;

cout << "Doctor Name: " << doc[i].name << endl;

cout << "Doctor Department: " << doc[i].department << endl;

cout << "Doctor Speciality: " << doc[i].speciality << endl;

cout << "Doctor Fees: " << doc[i].fees << endl;

cout << "Doctor Rating: " << doc[i].rate << endl;

cout << endl;

}

}

else

{

cout << endl;

cout << "No doc to display\n";

}

cout << endl;

}

else if (choice == 4)

{

cout << endl;

cout << "If you want to book any appointment, kindly visit the receptionist. THANK YOU." << endl;

cout << endl;

}

else if (choice == 5)

{

cout << endl;

string doctorNameOrSpeciality;

cout << "Enter the name or speciality of the doctor you want to rate: ";

cin >> doctorNameOrSpeciality;

bool visitedDoctor = false;

for (int i = 0; i < 50; i++)

{

if (pat[i].visit > 0 && (doctorNameOrSpeciality == doc[i].speciality || doctorNameOrSpeciality == doc[i].name)&& (pat[i].appointment=='y'||pat[i].appointment=='Y'))

{

visitedDoctor = true;

cout << "Add rating out of 5: ";

cin >> doc[i].rating;

cout << "Review added successfully for " << doc[i].name << "\n";

break;

}

}

if (!visitedDoctor)

{

cout << endl;

cout << "You haven't visited the specified doctor. Cannot add a review.\n";

}

cout << endl;

}

else

{

cout << endl;

cout << "invalid choice\n";

cout << endl;

}

}

int main() {

readData();

int choice;

do {

cout << "\n\n\n\n\n\t\t\t Loading... !\n";

char x = '\*';

for (int i = 0; i <= 27; i++)

{

Sleep(100);

cout << x;

}

system("cls");

cout << "\n\n\t\t\tWelcome to HOSPITAL!!" << endl;

cout << "HOSPITAL MANAGEMENT SYSTEM" << endl;

cout << "1.Admin" << endl;

cout << "2.Receptionist" << endl;

cout << "3.Doctor" << endl;

cout << "4.Patient" << endl;

cout << "5.Exit" << endl;

cout << "Enter your choice according to the list: ";

cin >> choice;

switch (choice)

{

case 1:

admin();

break;

case 2:

receptionist();

break;

case 3:

doctorFunc();

break;

case 4:

patientFunc();

break;

}

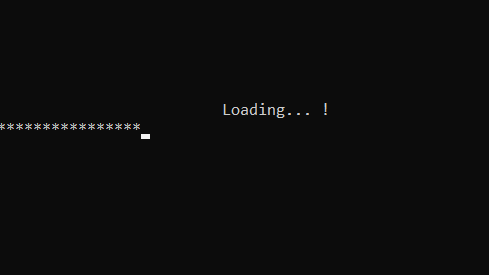
} while (choice != 5);

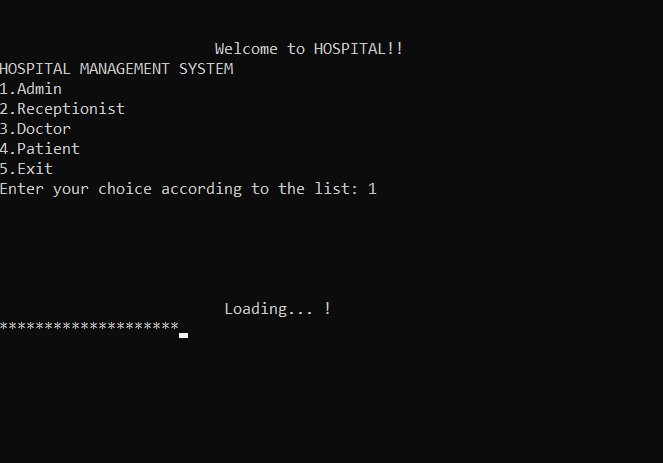
return 0;

}

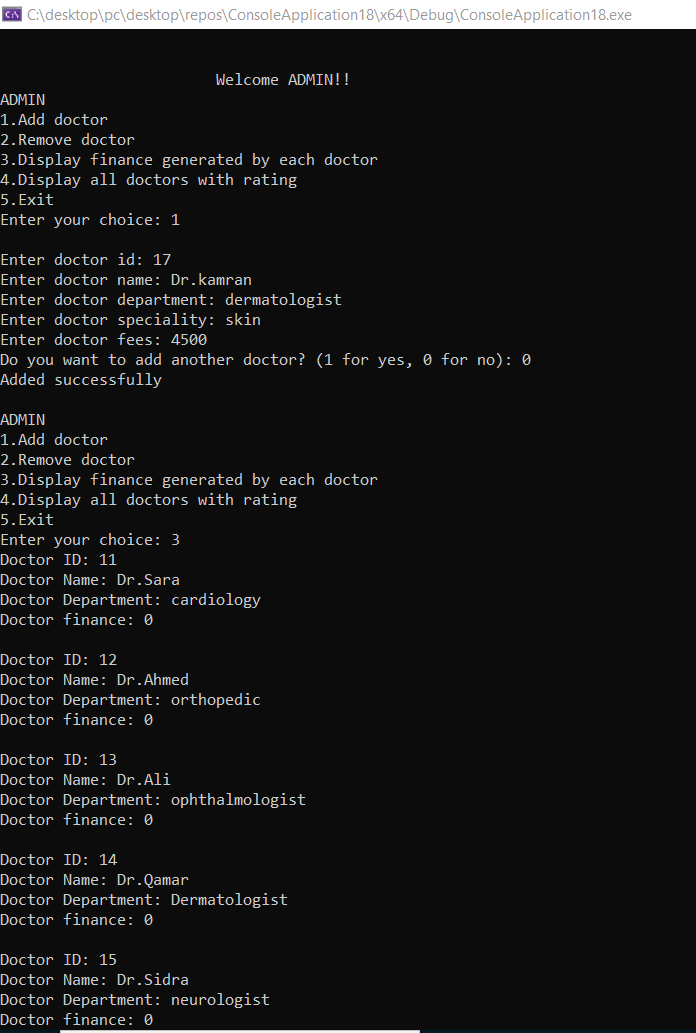
**Conclusion:**

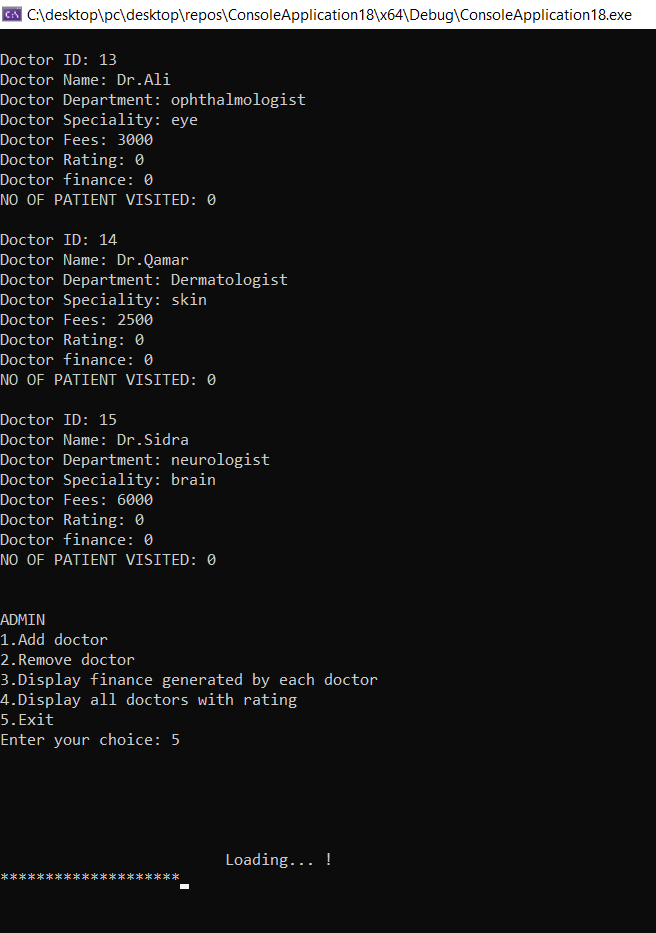
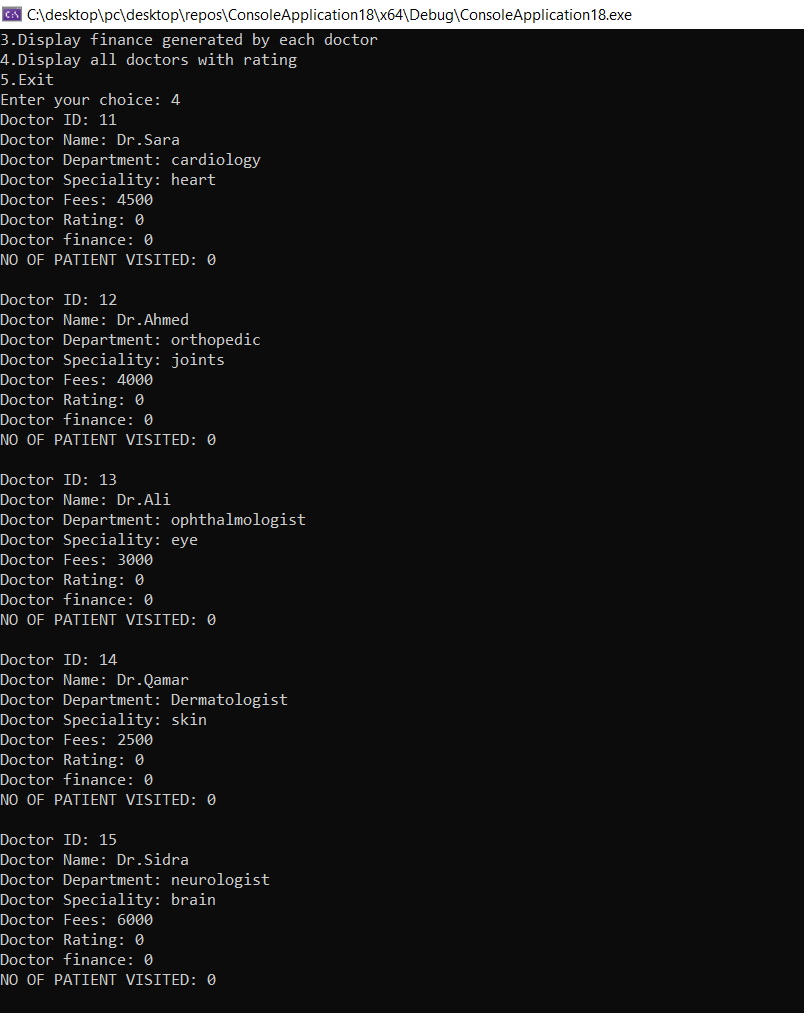
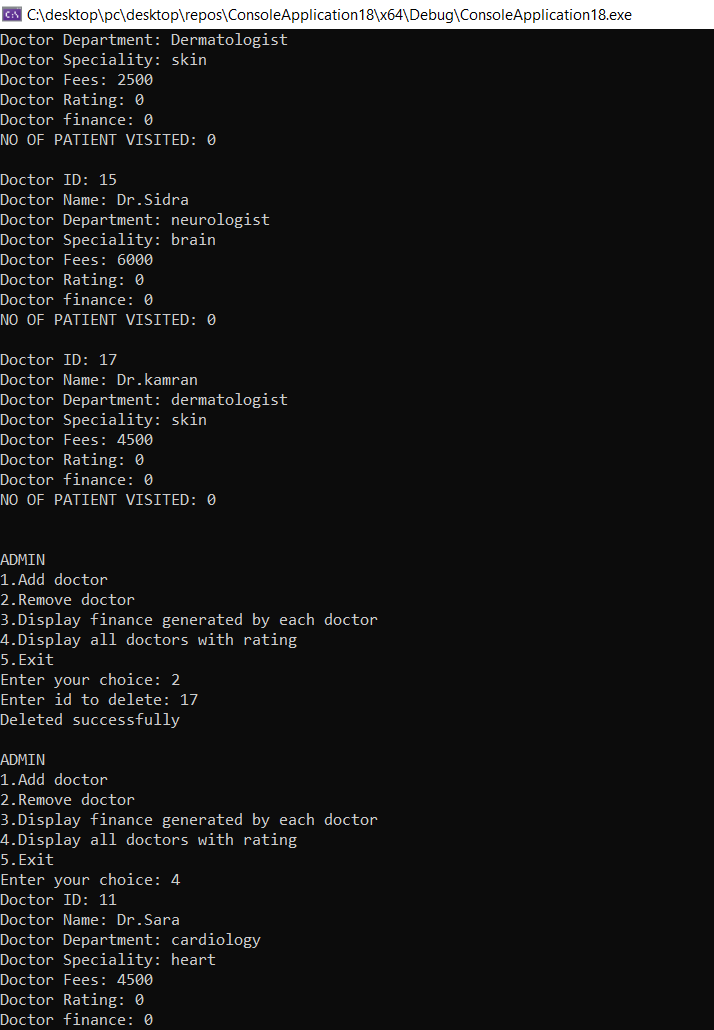
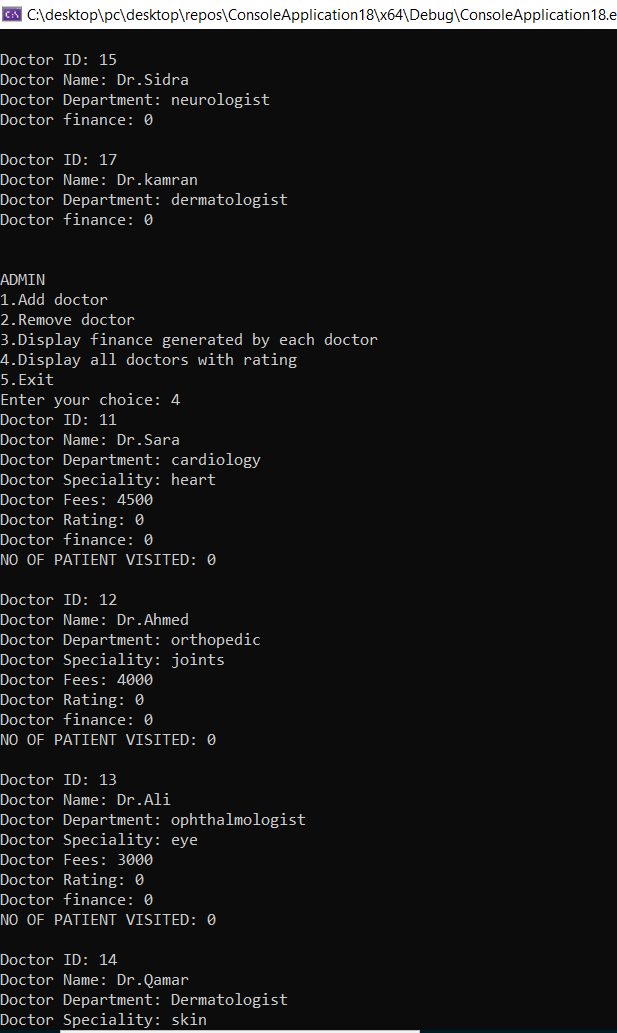
1. **Screenshots:**

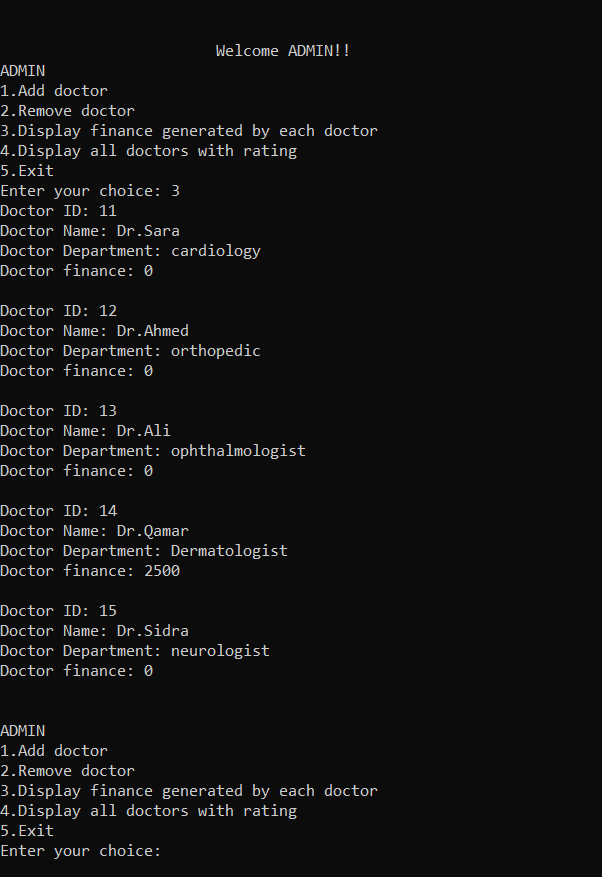


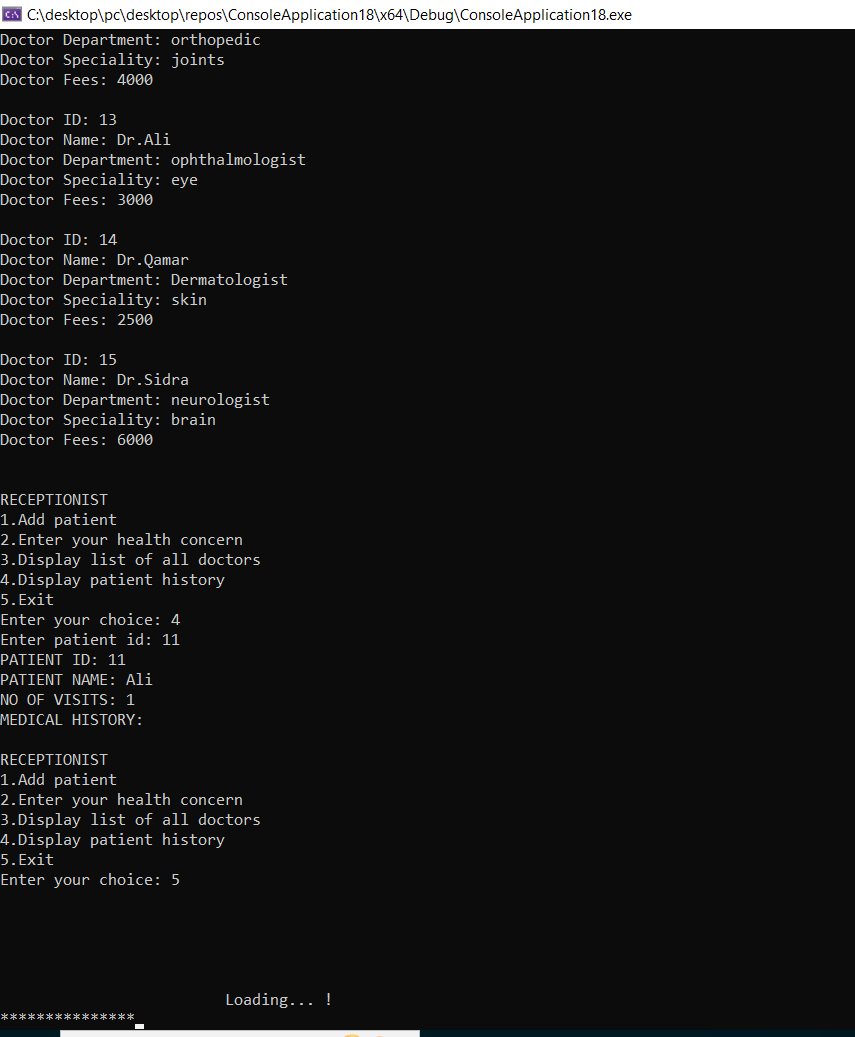
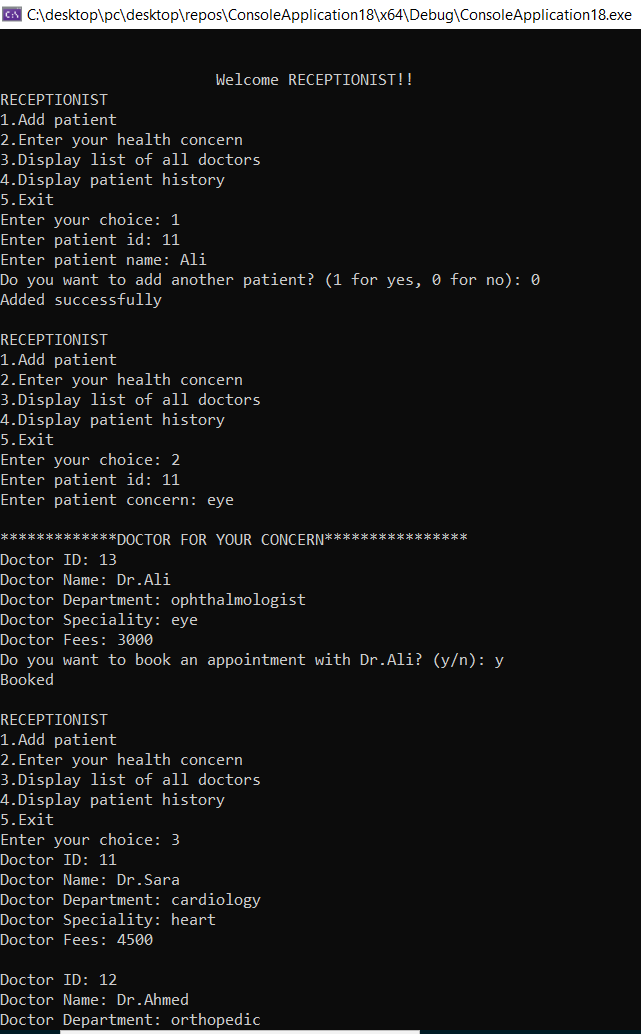


**ADMIN**

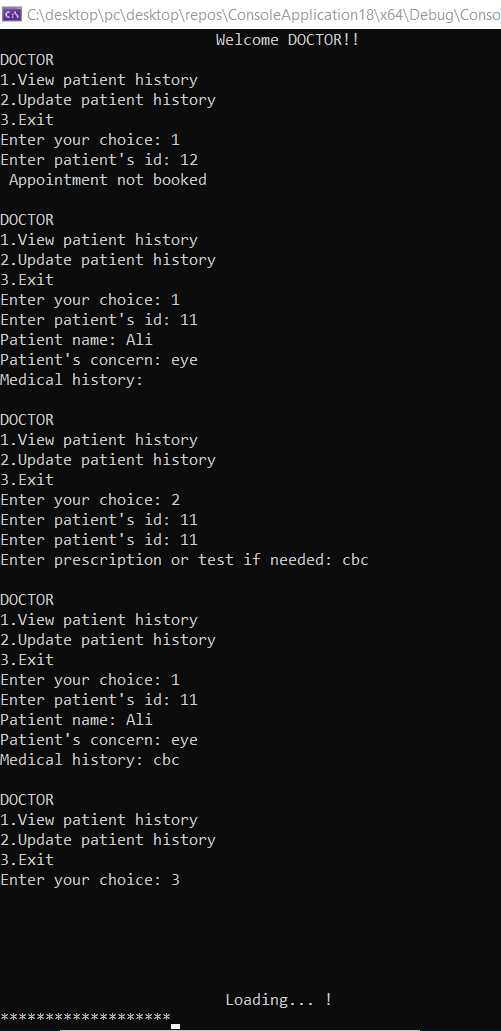


**//after patient booked the appointment**

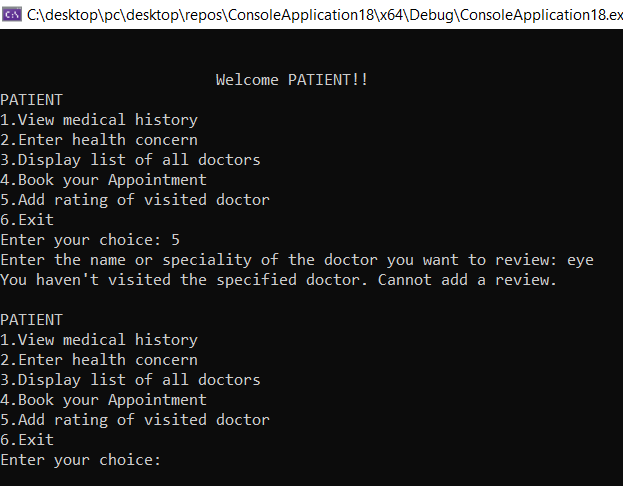
**RECEPTIONIST**

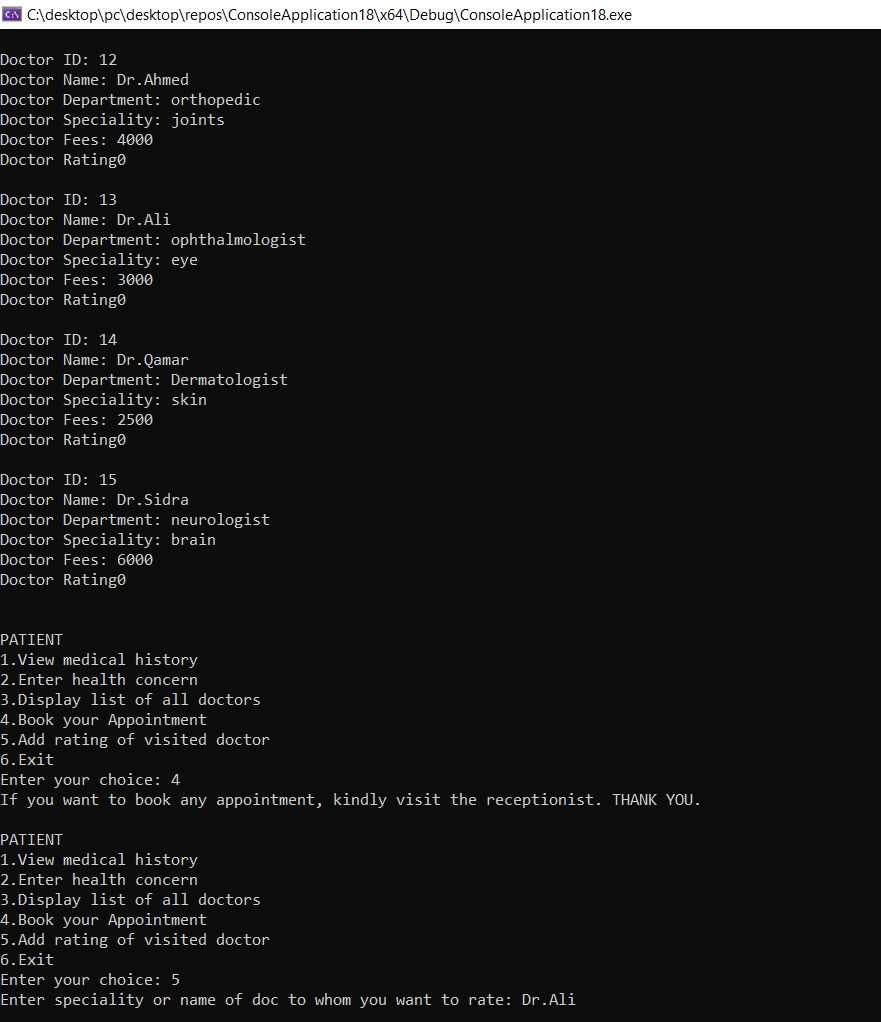
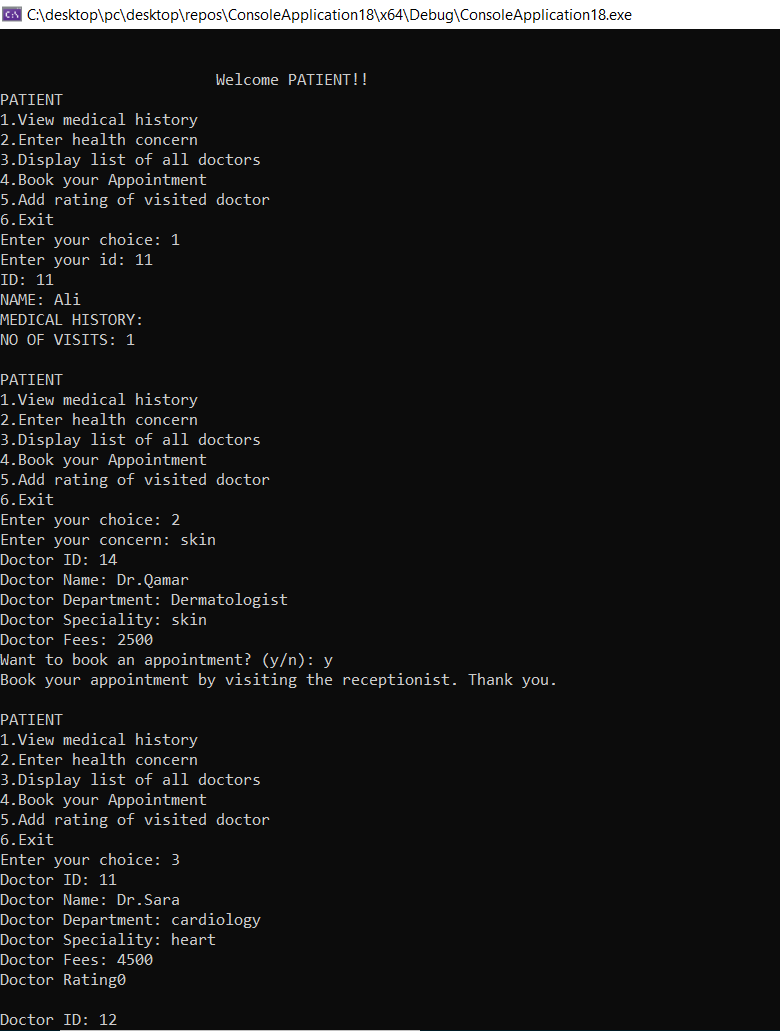


**DOCTOR**

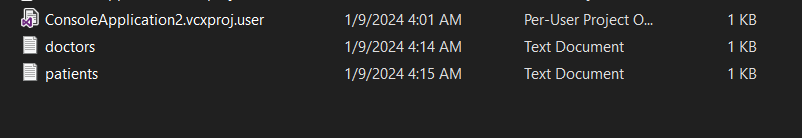


**PATIENT**





**File Handling:**

****

**Evaluation:**

**Objectives meet:**

* Ensure the accuracy and currency of patient records by implementing a system that allows for the regular update of patient information.
* Facilitate a user-friendly system that enables patients to choose their preferred doctor and book appointments efficiently.
* Develop a robust system for the management of doctors' information, including adding, updating, and deleting data as needed.
* Establish a feedback mechanism to gather patient opinions and experiences for continuous improvement.
* Efficiently manage the occupancy of rooms within the healthcare facility for various purposes.
* Establish a financial tracking system to monitor revenue generated from consultations in each department.