TO – DO LIST MANAGING APPLICATION AND HOSPITAL PATIENT RECORD SYSTEM USING C#

Submitted by

AMIYADEVI S
(Reg. No: 24MCR002)

JAYALAKSHMI S
(Reg. No: 24MCR042)

KEERTHANA R
(Reg. No: 24MCR056)

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

of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER APPLICATIONS

KONGUENGINEERING COLLEGE

(Autonomous)

PERUNDURAI, ERODE – 638 060

2025 -2026

TABLE OF CONTENTS

1. INTRODUCTION

This project document presents two C# console-based applications — a **To-Do List Managing Application** and a **Hospital Patient Record System**.Both are designed to demonstrate data management using CRUD (Create, Read, Update, Delete) operations. The To-Do List app focuses on personal task tracking and productivity, while the Hospital system emphasizes patient record maintenance and database connectivity using SQL Server.

2. OBJECTIVES

- To design and implement console-based C# applications that handle structured data efficiently.
- To understand CRUD operations using file storage (JSON) and relational databases (SQL Server).
- To develop user-friendly menu-driven interfaces for real-world task management.
- To apply object-oriented programming (OOP) concepts like classes, objects, methods, and encapsulation.
- To ensure data persistence and retrieval through serialization and database connections.

3. SYSTEM REQUIREMENTS

Hardware Requirements:

- Processor: Intel Core i3 or higher
- RAM: Minimum 4 GB
- Hard Disk: 500 MB free space
- Display: 1024×768 resolution

Software Requirements:

- Operating System: Windows 10 or later
- IDE: Visual Studio / Visual Studio Code
- Framework: .NET Framework 4.7+ or .NET Core

- Database: Microsoft SQL Server Express (for Hospital App)
- Libraries: `Newtonsoft.Json` (for To-Do List)

4. PROJECT OVERVIEW

4.1 To-Do List Managing Application

A console-based task manager that allows users to add, edit, view, delete, and search tasks. Each task includes a title, description, due date, category, and priority level. Data is saved in a `tasks.json` file for persistence between sessions.

Key Features:

- Add new tasks with details and priority.
- View all pending and completed tasks.
- Edit or delete existing tasks.
- Mark tasks as completed.
- Search or filter tasks by keyword, category, or status.

4.2 Hospital Patient Record System

A C# console application connected to an SQL Server database. It manages patient records such as name, gender, date of birth, phone, address, and medical history. It demonstrates backend database integration and CRUD operations using `System.Data.SqlClient`.

Key Features:

- Add, update, delete, and view patient information.
- Search patient records by name, phone, or blood group.
- Maintain audit fields like created and updated timestamps.
- Simple and interactive console-based UI.

5. MODULES DESCRIPTION

To-Do List Modules:

1. **Add Task Module:** Captures user input for task creation.
2. **Edit Task Module:** Allows modifying existing task details.
3. **View Task Module:** Displays all tasks with sorting and filters.
4. **Search Module:** Searches by keyword, category, or completion status.
5. **Storage Module:** Handles data persistence using JSON serialization.

Hospital App Modules:

1. **Database Module:** Creates and connects to the HospitalDB database.
2. **Patient Module:** Represents the patient object and properties.
3. **CRUD Module:** Performs insert, update, delete, and read operations using SQL queries.
4. **User Interface Module:** Console-based navigation and data entry for patients.

6. TECHNOLOGIES USED

- **Programming Language:** C# (.NET)
- **Database:** SQL Server 2019 (Hospital App)
- **File Storage:** JSON (To-Do List App)
- **Libraries:** Newtonsoft.Json for JSON serialization
- **IDE:** Visual Studio / Visual Studio Code
- **Data Access:** ADO.NET (SqlConnection, SqlCommand, SqlDataReader)

7. SYSTEM DESIGN

7.1 Architecture Overview

- The To-Do List follows a **file-based layered architecture**, where tasks are serialized to a JSON file.
- The Hospital App uses a **three-layer architecture**:
    - **Presentation Layer (Console UI)**
    - **Business Logic Layer (Patient.cs)**

- **Data Access Layer (Database.cs)**

7.2 Data Flow

User inputs → Application Logic → Database/File Storage → Output Display

7.3 Use Case Summary

- **Actors:** User (Administrator or Individual)
- **Use Cases:** Add, Edit, Delete, View, and Search records

8. IMPLEMENTATION

8.1 To-Do List Managing Application

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Newtonsoft.Json;  // Make sure to install via NuGet: Install-Package Newtonsoft.Json -Version 12.0.3
namespace ToDoListApp
{
    enum Priority { Low, Medium, High }
    class TaskItem
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public DateTime DueDate { get; set; }
        public Priority TaskPriority { get; set; }
```

```
    public string Category { get; set; }
    public bool IsCompleted { get; set; }
    public override string ToString()
    {
        string status = IsCompleted ? "[✓ Completed]" : "[ ] Pending";
        return $"ID: {Id} | {Title} | Due: {DueDate.ToShortDateString()} | Priority: {TaskPriority} |
Category: {Category} | {status}\n    {Description}";
    }
  }
  class Program
  {
    static List<TaskItem> tasks = new List<TaskItem>();
    static int nextId = 1;
    static string filePath = "tasks.json";
    static void Main(string[] args)
    {
        Console.Title = "Formal To-Do List Managing Application";
        LoadTasks();
        bool running = true;
        while (running)
        {
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("=======================================");
            Console.WriteLine("       TO-DO LIST APPLICATION");
            Console.WriteLine("=======================================");
```

```
Console.ResetColor();
Console.WriteLine("1. Add Task");
Console.WriteLine("2. View All Tasks");
Console.WriteLine("3. Edit Task");
Console.WriteLine("4. Mark Task as Completed");
Console.WriteLine("5. Delete Task");
Console.WriteLine("6. Search / Filter Tasks");
Console.WriteLine("7. Save & Exit");
Console.WriteLine("=======================================");
Console.Write("Enter your choice: ");
string choice = Console.ReadLine();
switch (choice)
{
   case "1": AddTask(); break;
   case "2": ViewTasks(); break;
   case "3": EditTask(); break;
   case "4": MarkTaskCompleted(); break;
   case "5": DeleteTask(); break;
   case "6": SearchTasks(); break;
   case "7": SaveTasks(); running = false; break;
   default: Console.WriteLine("Invalid choice. Try again!"); break;
}
if (running)
{
   Console.WriteLine("\nPress any key to return to menu...");
   Console.ReadKey();
```

```
        }
    }
}
static void AddTask()
{
    Console.Clear();
    Console.Write("Enter Task Title: ");
    string title = Console.ReadLine();
    Console.Write("Enter Description: ");
    string desc = Console.ReadLine();
    Console.Write("Enter Due Date (yyyy-mm-dd): ");
    DateTime dueDate;
    while (!DateTime.TryParse(Console.ReadLine(), out dueDate))
        Console.Write("Invalid date, try again: ");
    Console.Write("Enter Priority (Low/Medium/High): ");
    Priority priority;
    while (!Enum.TryParse(Console.ReadLine(), true, out priority))
        Console.Write("Invalid input, enter Low/Medium/High: ");
    Console.Write("Enter Category (Work/Personal/Study/etc): ");
    string category = Console.ReadLine();
    TaskItem task = new TaskItem
    {
        Id = nextId++,
        Title = title,
        Description = desc,
        DueDate = dueDate,
```

```
        TaskPriority = priority,
        Category = string.IsNullOrWhiteSpace(category) ? "General" : category,
        IsCompleted = false
    };
    tasks.Add(task);
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("Task added successfully!");
    Console.ResetColor();
}
static void ViewTasks()
{
    Console.Clear();
    Console.WriteLine("========= ALL TASKS =========");
    if (tasks.Count == 0)
    {
        Console.WriteLine("No tasks available!");
        return;
    }
    var sortedTasks = tasks.OrderBy(t => t.DueDate).ThenByDescending(t => t.TaskPriority);
    foreach (var task in sortedTasks)
    {
        Console.WriteLine(task);
        Console.WriteLine("---------------------------------");
    }
}
static void EditTask()
```

```
    {
        Console.Clear();
        Console.Write("Enter Task ID to edit: ");
        int id;
        if (int.TryParse(Console.ReadLine(), out id))
        {
            TaskItem task = tasks.FirstOrDefault(t => t.Id == id);
            if (task != null)
            {
                Console.Write("New Title (leave blank to keep): ");
                string newTitle = Console.ReadLine();
                if (!string.IsNullOrWhiteSpace(newTitle)) task.Title = newTitle;
                Console.Write("New Description (leave blank to keep): ");
                string newDesc = Console.ReadLine();
                if (!string.IsNullOrWhiteSpace(newDesc)) task.Description = newDesc;
                Console.Write("New Due Date (leave blank to keep): ");
                string dateInput = Console.ReadLine();
                DateTime newDate;
                if (!string.IsNullOrWhiteSpace(dateInput) && DateTime.TryParse(dateInput, out
newDate))
                    task.DueDate = newDate;
                Console.Write("New Priority (Low/Medium/High, leave blank to keep): ");
                string priInput = Console.ReadLine();
                Priority newPriority;
                if (!string.IsNullOrWhiteSpace(priInput) && Enum.TryParse(priInput, true, out
newPriority))
                    task.TaskPriority = newPriority;
```

```
            Console.Write("New Category (leave blank to keep): ");
            string newCat = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(newCat)) task.Category = newCat;
            Console.WriteLine("Task updated successfully!");
        }
        else Console.WriteLine("Task not found!");
    }
    else Console.WriteLine("Invalid ID!");
}
static void MarkTaskCompleted()
{
    Console.Clear();
    Console.Write("Enter Task ID to mark as completed: ");
    int id;
    if (int.TryParse(Console.ReadLine(), out id))
    {
        TaskItem task = tasks.FirstOrDefault(t => t.Id == id);
        if (task != null)
        {
            task.IsCompleted = true;
            Console.WriteLine("Task marked as completed!");
        }
        else Console.WriteLine("Task not found!");
    }
    else Console.WriteLine("Invalid ID!");
}
```

```
static void DeleteTask()
{
    Console.Clear();
    Console.Write("Enter Task ID to delete: ");
    int id;
    if (int.TryParse(Console.ReadLine(), out id))
    {
        TaskItem task = tasks.FirstOrDefault(t => t.Id == id);
        if (task != null)
        {
            tasks.Remove(task);
            Console.WriteLine("Task deleted successfully!");
        }
        else Console.WriteLine("Task not found!");
    }
    else Console.WriteLine("Invalid ID!");
}
static void SearchTasks()
{
    Console.Clear();
    Console.WriteLine("Search Options:");
    Console.WriteLine("1. By Keyword");
    Console.WriteLine("2. By Status");
    Console.WriteLine("3. By Category");
    Console.Write("Choose option: ");
    string choice = Console.ReadLine();
```

```
        IEnumerable<TaskItem> result = tasks;
        switch (choice)
        {
            case "1":
                Console.Write("Enter keyword: ");
                string keyword = Console.ReadLine().ToLower();
                result = tasks.Where(t => t.Title.ToLower().Contains(keyword) ||
t.Description.ToLower().Contains(keyword));
                break;
            case "2":
                Console.Write("Enter status (Completed/Pending): ");
                string status = Console.ReadLine().ToLower();
                if (status == "completed")
                    result = tasks.Where(t => t.IsCompleted);
                else if (status == "pending")
                    result = tasks.Where(t => !t.IsCompleted);
                else
                {
                    Console.WriteLine("Invalid status!");
                    return;
                }
                break;
            case "3":
                Console.Write("Enter category: ");
                string category = Console.ReadLine().ToLower();
```

```
            result = tasks.Where(t => t.Category.ToLower() == category);
            break;
        default:
            Console.WriteLine("Invalid choice!");
            return;
    }
    Console.WriteLine("========= SEARCH RESULTS =========");
    foreach (var task in result)
    {
        Console.WriteLine(task);
        Console.WriteLine("-----------------------------------");
    }
}
static void SaveTasks()
{
    File.WriteAllText(filePath, JsonConvert.SerializeObject(tasks, Formatting.Indented));
    Console.WriteLine("Tasks saved successfully!");
}
static void LoadTasks()
{
    if (File.Exists(filePath))
    {
        string data = File.ReadAllText(filePath);
        tasks = JsonConvert.DeserializeObject<List<TaskItem>>(data) ?? new List<TaskItem>();
        if (tasks.Count > 0)
            nextId = tasks.Max(t => t.Id) + 1;
```

```
        }
      }
   }
}
```

8.2 Hospital Patient Record System

QUERY

```
IF DB_ID('HospitalDB') IS NULL
   CREATE DATABASE HospitalDB;
GO
USE HospitalDB;
GO
IF OBJECT_ID('dbo.Patients','U') IS NOT NULL
   DROP TABLE dbo.Patients;
GO
CREATE TABLE dbo.Patients
(
  PatientId INT IDENTITY(1,1) PRIMARY KEY,
  FirstName NVARCHAR(100) NOT NULL,
  LastName NVARCHAR(100) NULL,
  Gender CHAR(1) NULL,
  DateOfBirth DATE NULL,
  Phone NVARCHAR(20) NULL,
  Address NVARCHAR(250) NULL,
  BloodGroup NVARCHAR(5) NULL,
  MedicalHistory NVARCHAR(MAX) NULL,
  CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),
```

```
  UpdatedAt DATETIME NULL
);
GO
INSERT INTO dbo.Patients(FirstName, LastName, Gender, DateOfBirth, Phone, Address, BloodGroup, MedicalHistory)
VALUES
('Arun','Kumar','M','1988-05-12','9876543210','Chennai','B+','Hypertension'),
('Meena','R','F','1994-11-03','9123456780','Madurai','O+','None');
GO
```

CONSOLE APP

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="HospitalDb" connectionString="Data Source=SAKTHIVELU\SQLEXPRESS;Initial Catalog=HospitalDB;Integrated Security=True;Connect Timeout=30;"
      providerName="System.Data.SqlClient" />
    <add name="HospitalApp.Properties.Settings.HospitalDBConnectionString"
      connectionString="Data Source=SAKTHIVELU\SQLEXPRESS;Initial Catalog=HospitalDB;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

C# FILES

Patient.cs

```
using System;

namespace HospitalConsoleApp
{
    public class Patient
    {
        public int PatientId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public char? Gender { get; set; }
        public DateTime? DateOfBirth { get; set; }
        public string Phone { get; set; }
        public string Address { get; set; }
        public string BloodGroup { get; set; }
        public string MedicalHistory { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime? UpdatedAt { get; set; }
        public override string ToString()
        {
            return $"[{PatientId}] {FirstName} {LastName} | Gender: {Gender} | DOB:
{(DateOfBirth?.ToString("yyyy-MM-dd") ?? "N/A")} | Phone: {Phone} | Blood:
{BloodGroup}";
        }
    }
}
```

Database.cs (Data Access Layer)

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
namespace HospitalConsoleApp
{
    public class Database
    {
        private readonly string _connString;


        public Database()
        {
            _connString =
ConfigurationManager.ConnectionStrings["HospitalDb"].ConnectionString;
        }
        private SqlConnection GetConnection()
        {
            return new SqlConnection(_connString);
        }
        public int AddPatient(Patient p)
        {
            string sql = @"
                INSERT INTO dbo.Patients
                (FirstName, LastName, Gender, DateOfBirth, Phone, Address, BloodGroup,
MedicalHistory)
                VALUES
```

```
            (@FirstName, @LastName, @Gender, @DateOfBirth, @Phone, @Address,
@BloodGroup, @MedicalHistory);
            SELECT SCOPE_IDENTITY();
        ";
        using (var conn = GetConnection())
        using (var cmd = new SqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@FirstName", p.FirstName);
            cmd.Parameters.AddWithValue("@LastName", (object)p.LastName ?? DBNull.Value);
            cmd.Parameters.AddWithValue("@Gender", (object)(p.Gender.HasValue ?
p.Gender.Value.ToString() : (object)DBNull.Value));
            cmd.Parameters.AddWithValue("@DateOfBirth", (object)p.DateOfBirth ??
DBNull.Value);
            cmd.Parameters.AddWithValue("@Phone", (object)p.Phone ?? DBNull.Value);
            cmd.Parameters.AddWithValue("@Address", (object)p.Address ?? DBNull.Value);
            cmd.Parameters.AddWithValue("@BloodGroup", (object)p.BloodGroup ??
DBNull.Value);
            cmd.Parameters.AddWithValue("@MedicalHistory", (object)p.MedicalHistory ??
DBNull.Value);
            conn.Open();
            var idObj = cmd.ExecuteScalar();
            return Convert.ToInt32(idObj);
        }
    }
    public bool UpdatePatient(Patient p)
    {
        string sql = @"
            UPDATE dbo.Patients SET
```

```
            FirstName=@FirstName, LastName=@LastName, Gender=@Gender,
DateOfBirth=@DateOfBirth,
            Phone=@Phone, Address=@Address, BloodGroup=@BloodGroup,
MedicalHistory=@MedicalHistory,
            UpdatedAt=GETDATE()
          WHERE PatientId=@PatientId;
        ";
        using (var conn = GetConnection())
        using (var cmd = new SqlCommand(sql, conn))
        {
          cmd.Parameters.AddWithValue("@PatientId", p.PatientId);
          cmd.Parameters.AddWithValue("@FirstName", p.FirstName);
          cmd.Parameters.AddWithValue("@LastName", (object)p.LastName ?? DBNull.Value);
          cmd.Parameters.AddWithValue("@Gender", (object)(p.Gender.HasValue ?
p.Gender.Value.ToString() : (object)DBNull.Value));
          cmd.Parameters.AddWithValue("@DateOfBirth", (object)p.DateOfBirth ??
DBNull.Value);
          cmd.Parameters.AddWithValue("@Phone", (object)p.Phone ?? DBNull.Value);
          cmd.Parameters.AddWithValue("@Address", (object)p.Address ?? DBNull.Value);
          cmd.Parameters.AddWithValue("@BloodGroup", (object)p.BloodGroup ??
DBNull.Value);
          cmd.Parameters.AddWithValue("@MedicalHistory", (object)p.MedicalHistory ??
DBNull.Value);
          conn.Open();
          int rows = cmd.ExecuteNonQuery();
          return rows > 0;
        }
      }
```

```
public bool DeletePatient(int patientId)
{
    string sql = "DELETE FROM dbo.Patients WHERE PatientId = @PatientId";
    using (var conn = GetConnection())
    using (var cmd = new SqlCommand(sql, conn))
    {
        cmd.Parameters.AddWithValue("@PatientId", patientId);
        conn.Open();
        int rows = cmd.ExecuteNonQuery();
        return rows > 0;
    }
}
public Patient GetPatientById(int patientId)
{
    string sql = "SELECT * FROM dbo.Patients WHERE PatientId = @PatientId";
    using (var conn = GetConnection())
    using (var cmd = new SqlCommand(sql, conn))
    {
        cmd.Parameters.AddWithValue("@PatientId", patientId);
        conn.Open();
        using (var r = cmd.ExecuteReader())
        {
            if (r.Read())
                return ReadPatient(r);
        }
    }
    return null;
```

```
    }
    public List<Patient> GetAllPatients()
    {
        var list = new List<Patient>();
        string sql = "SELECT * FROM dbo.Patients ORDER BY PatientId";
        using (var conn = GetConnection())
        using (var cmd = new SqlCommand(sql, conn))
        {
            conn.Open();
            using (var r = cmd.ExecuteReader())
            {
                while (r.Read())
                {
                    list.Add(ReadPatient(r));
                }
            }
        }
        return list;
    }
    public List<Patient> SearchPatients(string term)
    {
        var list = new List<Patient>();
        string sql = @"SELECT * FROM dbo.Patients
                       WHERE FirstName LIKE @t OR LastName LIKE @t OR Phone LIKE @t OR
BloodGroup LIKE @t
                       ORDER BY PatientId";
        using (var conn = GetConnection())
```

```
        using (var cmd = new SqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@t", "%" + term + "%");
            conn.Open();
            using (var r = cmd.ExecuteReader())
            {
                while (r.Read())
                {
                    list.Add(ReadPatient(r));
                }
            }
        }
        return list;
    }
    private Patient ReadPatient(SqlDataReader r)
    {
        return new Patient
        {
            PatientId = Convert.ToInt32(r["PatientId"]),
            FirstName = r["FirstName"] as string,
            LastName = r["LastName"] as string,
            Gender = r["Gender"] == DBNull.Value ? (char?)null : Convert.ToChar(r["Gender"]),
            DateOfBirth = r["DateOfBirth"] == DBNull.Value ? (DateTime?)null :
Convert.ToDateTime(r["DateOfBirth"]),
            Phone = r["Phone"] as string,
            Address = r["Address"] as string,
            BloodGroup = r["BloodGroup"] as string,
```

```
            MedicalHistory = r["MedicalHistory"] as string,
            CreatedAt = Convert.ToDateTime(r["CreatedAt"]),
            UpdatedAt = r["UpdatedAt"] == DBNull.Value ? (DateTime?)null :
Convert.ToDateTime(r["UpdatedAt"])
        };
    }
  }
}
```

Program.cs (Console UI + CRUD menu)

```
using System;
using System.Collections.Generic;
using System.Globalization;
namespace HospitalConsoleApp
{
  class Program
  {
    static Database db = new Database();

    static void Main(string[] args)
    {
      Console.Title = "Hospital Patient Record System - Console";
      RunMenu();
    }

    static void RunMenu()
    {
      while (true)
```

```
{
    Console.Clear();
    Console.WriteLine("=== Hospital Patient Record System ===");
    Console.WriteLine("1. Add new patient");
    Console.WriteLine("2. Update patient");
    Console.WriteLine("3. Delete patient");
    Console.WriteLine("4. View patient by ID");
    Console.WriteLine("5. List all patients");
    Console.WriteLine("6. Search patients");
    Console.WriteLine("0. Exit");
    Console.Write("Choose option: ");
    var key = Console.ReadLine();
    switch (key)
    {
        case "1": AddPatientFlow(); break;
        case "2": UpdatePatientFlow(); break;
        case "3": DeletePatientFlow(); break;
        case "4": ViewByIdFlow(); break;
        case "5": ListAllFlow(); break;
        case "6": SearchFlow(); break;
        case "0": return;
        default:
            Console.WriteLine("Invalid option. Press Enter.");
            Console.ReadLine();
            break;
    }
}
```

```
}
static void AddPatientFlow()
{
    Console.Clear();
    Console.WriteLine("--- Add Patient ---");
    var p = ReadPatientFromConsole(new Patient());
    try
    {
        int newId = db.AddPatient(p);
        Console.WriteLine("Patient added with ID: " + newId);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error adding patient: " + ex.Message);
    }
    Pause();
}
static void UpdatePatientFlow()
{
    Console.Clear();
    Console.Write("Enter Patient ID to update: ");
    int id;
    string input = Console.ReadLine();
    if (!int.TryParse(input, out id))
    {
        Console.WriteLine("Invalid id");
        Pause(); return;
```

```
    }
    var existing = db.GetPatientById(id);
    if (existing == null)
    {
        Console.WriteLine("Patient not found");
        Pause(); return;
    }
    Console.WriteLine("Leave blank to keep current value.");
    var updated = ReadPatientFromConsole(existing);
    updated.PatientId = id;
    try
    {
        bool ok = db.UpdatePatient(updated);
        Console.WriteLine(ok ? "Updated successfully." : "Update failed.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error updating: " + ex.Message);
    }
    Pause();
}
static void DeletePatientFlow()
{
    Console.Clear();
    Console.Write("Enter Patient ID to delete: ");
    int id;
    string input = Console.ReadLine();
```

```
    if (!int.TryParse(input, out id))
    {
        Console.WriteLine("Invalid id");
        Pause(); return;
    }
    Console.Write("Are you sure? (y/n): ");
    if (Console.ReadLine().ToLower() != "y")
    {
        Console.WriteLine("Cancelled.");
        Pause(); return;
    }
    try
    {
        bool ok = db.DeletePatient(id);
        Console.WriteLine(ok ? "Deleted." : "Not found / not deleted.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error deleting: " + ex.Message);
    }
    Pause();
}
static void ViewByIdFlow()
{
    Console.Clear();
    Console.Write("Enter Patient ID: ");
    int id;
```

```
    string input = Console.ReadLine();
    if (!int.TryParse(input, out id))
    {
        Console.WriteLine("Invalid id");
        Pause(); return;
    }
    var p = db.GetPatientById(id);
    if (p == null) Console.WriteLine("Not found.");
    else ShowPatientDetails(p);
    Pause();
}
static void ListAllFlow()
{
    Console.Clear();
    Console.WriteLine("--- All Patients ---");
    var list = db.GetAllPatients();
    foreach (var p in list)
    {
        Console.WriteLine(p.ToString());
    }
    Console.WriteLine("Total: " + list.Count);
    Pause();
}
static void SearchFlow()
{
    Console.Clear();
    Console.Write("Search term (name/phone/blood): ");
```

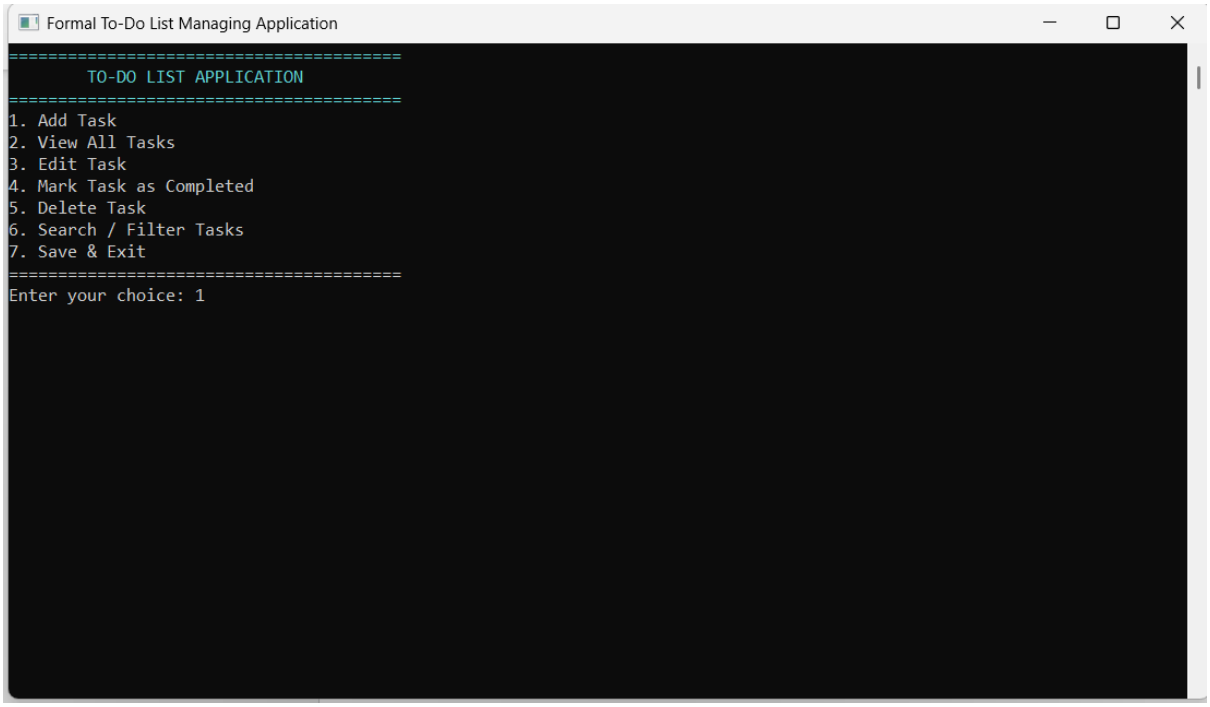
```
            string term = Console.ReadLine();
            var list = db.SearchPatients(term);
            Console.WriteLine("Matches: " + list.Count);
            foreach (var p in list) Console.WriteLine(p.ToString());
            Pause();
        }
        static Patient ReadPatientFromConsole(Patient current)
        {
            var p = new Patient();
            string s;
            Console.Write("First name [" + current.FirstName + "]: ");
            s = Console.ReadLine();
            p.FirstName = string.IsNullOrWhiteSpace(s) ? current.FirstName : s.Trim();
            Console.Write("Last name [" + current.LastName + "]: ");
            s = Console.ReadLine();
            p.LastName = string.IsNullOrWhiteSpace(s) ? current.LastName : s.Trim();
            Console.Write("Gender (M/F/O) [" + (current.Gender.HasValue ?
current.Gender.ToString() : "") + "]: ");
            s = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(s)) p.Gender = current.Gender;
            else p.Gender = s.Trim().Length > 0 ? (char?)char.ToUpperInvariant(s.Trim()[0]) : null;
            Console.Write("Date of Birth (yyyy-MM-dd) [" + (current.DateOfBirth.HasValue ?
current.DateOfBirth.Value.ToString("yyyy-MM-dd") : "") + "]: ");
            s = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(s)) p.DateOfBirth = current.DateOfBirth;
            else
            {
                DateTime dob;
```

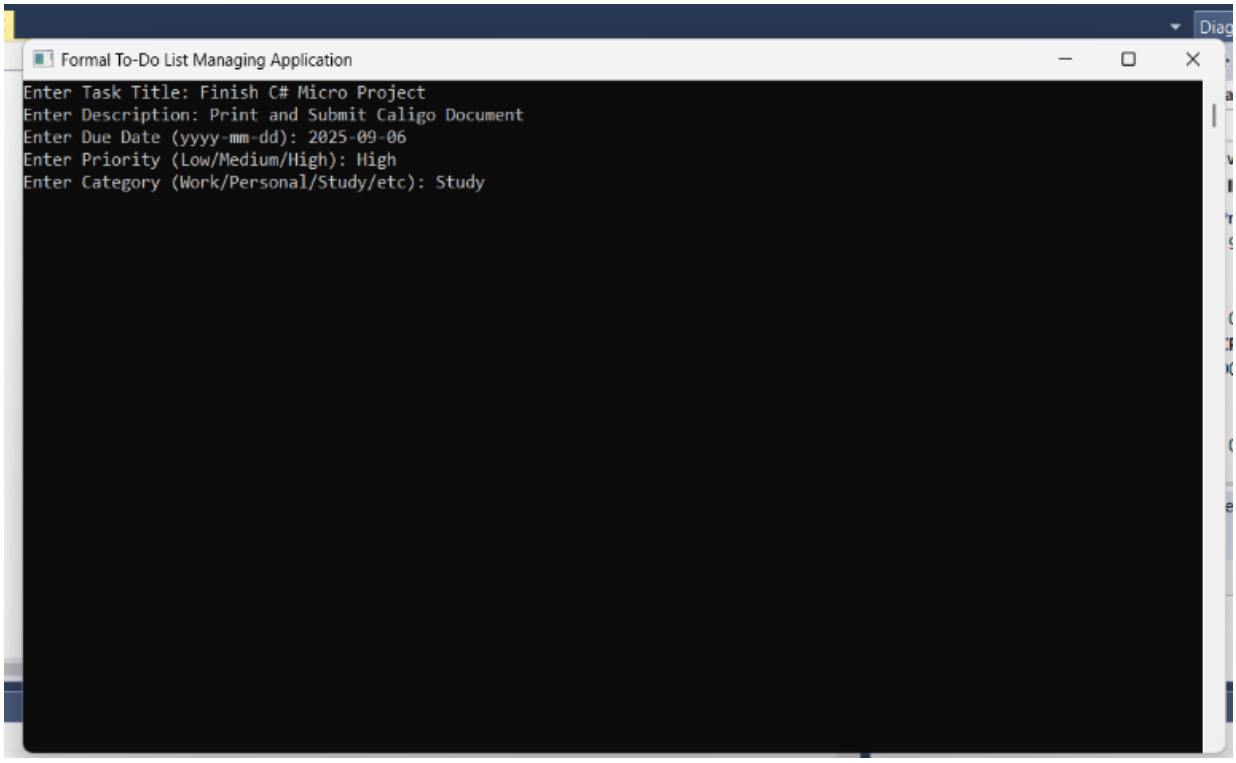
```
            if (DateTime.TryParseExact(s.Trim(), "yyyy-MM-dd", CultureInfo.InvariantCulture,
DateTimeStyles.None, out dob))
                p.DateOfBirth = dob;
            else
                p.DateOfBirth = current.DateOfBirth;
        }

        Console.Write("Phone [" + current.Phone + "]: ");
        s = Console.ReadLine();
        p.Phone = string.IsNullOrWhiteSpace(s) ? current.Phone : s.Trim();

        Console.Write("Address [" + current.Address + "]: ");
        s = Console.ReadLine();
        p.Address = string.IsNullOrWhiteSpace(s) ? current.Address : s.Trim();

        Console.Write("Blood Group [" + current.BloodGroup + "]: ");
        s = Console.ReadLine();
        p.BloodGroup = string.IsNullOrWhiteSpace(s) ? current.BloodGroup : s.Trim();

        Console.Write("Medical History [" + current.MedicalHistory + "]: ");
        s = Console.ReadLine();
        p.MedicalHistory = string.IsNullOrWhiteSpace(s) ? current.MedicalHistory : s.Trim();
        return p;
    }
    static void ShowPatientDetails(Patient p)
    {
        Console.WriteLine("------ Patient Detail ------");
```

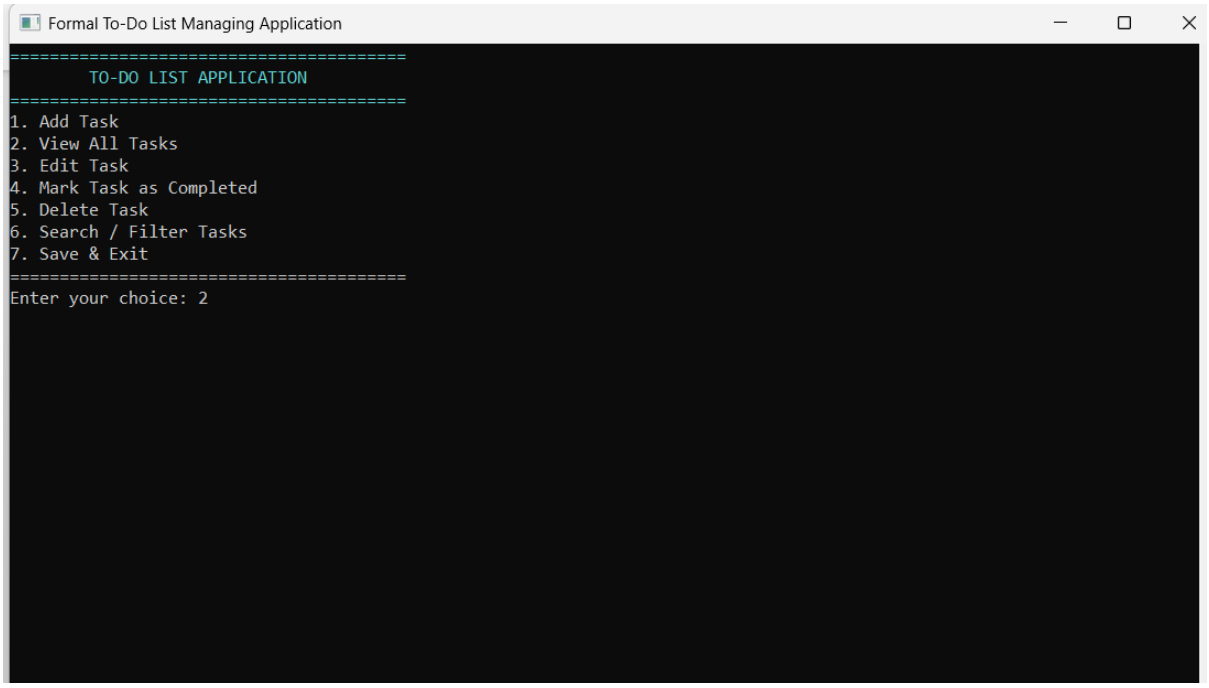
```
            Console.WriteLine("ID: " + p.PatientId);
            Console.WriteLine("Name: " + p.FirstName + " " + p.LastName);
            Console.WriteLine("Gender: " + (p.Gender.HasValue ? p.Gender.ToString() : "N/A"));
            Console.WriteLine("DOB: " + (p.DateOfBirth.HasValue ?
p.DateOfBirth.Value.ToString("yyyy-MM-dd") : "N/A"));
            Console.WriteLine("Phone: " + p.Phone);
            Console.WriteLine("Address: " + p.Address);
            Console.WriteLine("Blood Group: " + p.BloodGroup);
            Console.WriteLine("Medical History: " + p.MedicalHistory);
            Console.WriteLine("CreatedAt: " + p.CreatedAt);
            Console.WriteLine("UpdatedAt: " + (p.UpdatedAt.HasValue ?
p.UpdatedAt.Value.ToString() : "N/A"));
        }

        static void Pause()
        {
            Console.WriteLine();
            Console.Write("Press Enter to continue...");
            Console.ReadLine();
        }
    }
}
```

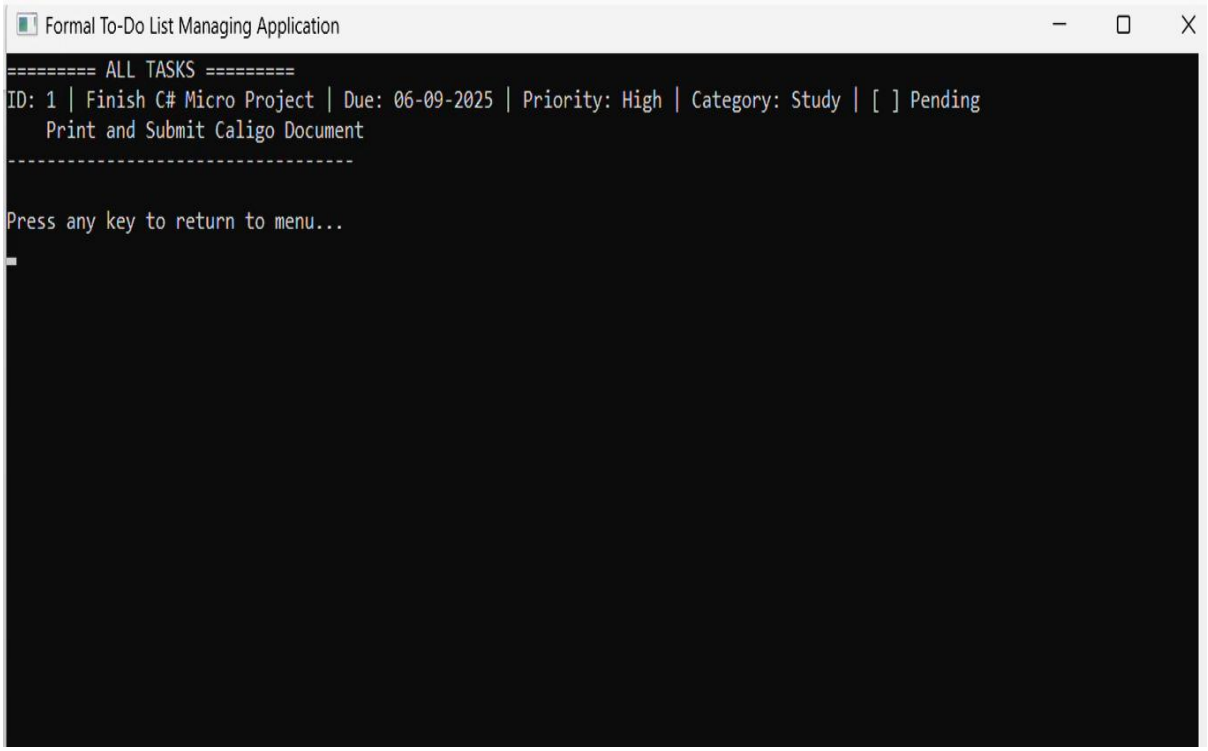
9. OUTPUT SCREENS

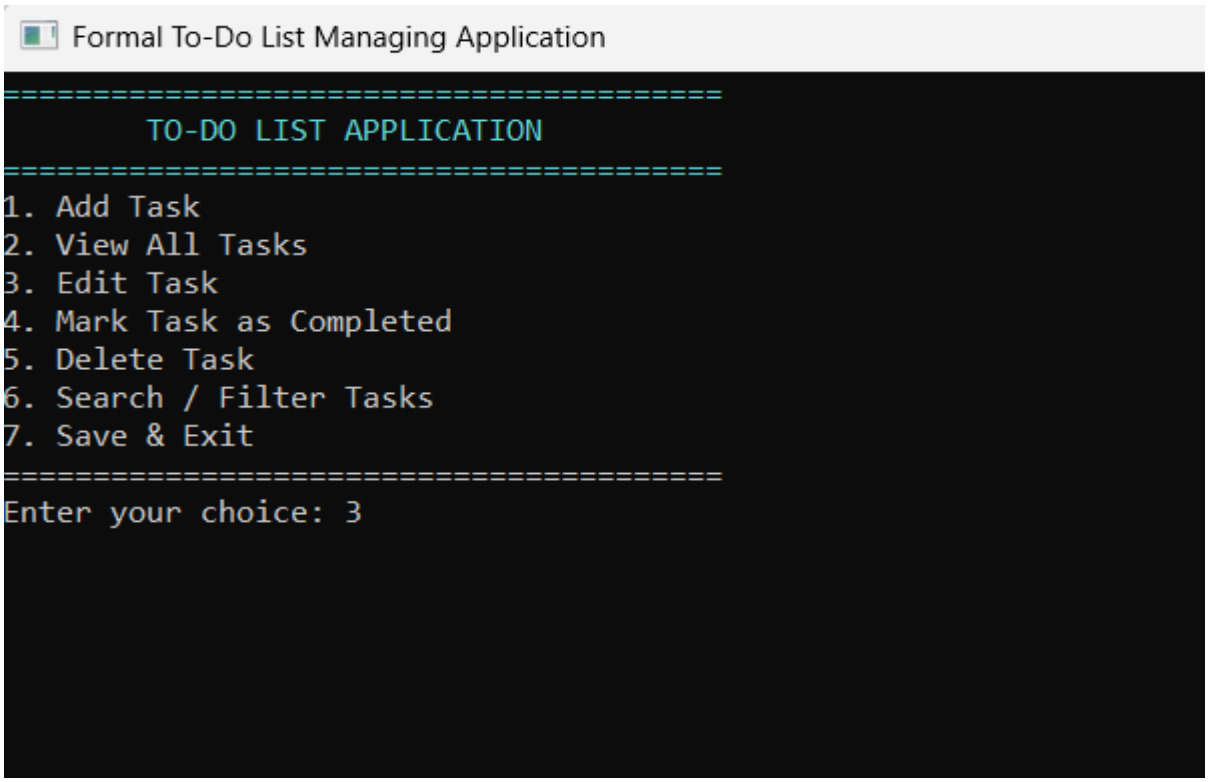
To-Do List Application:

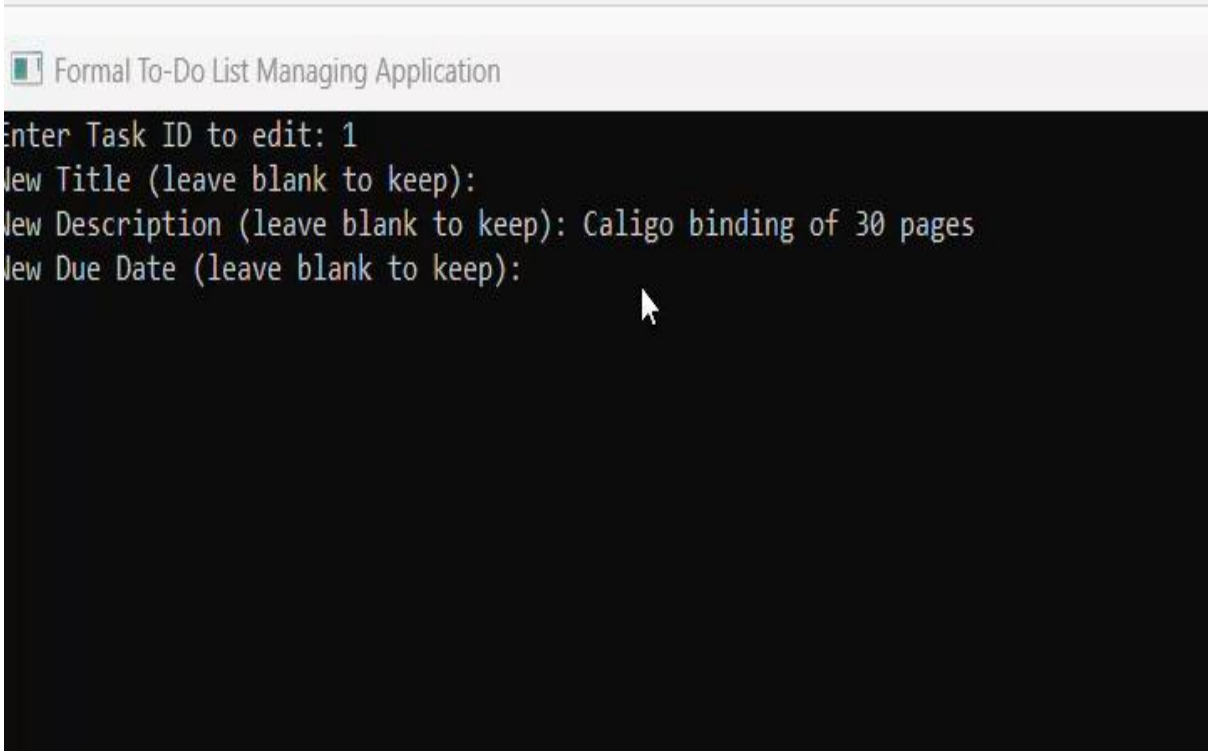


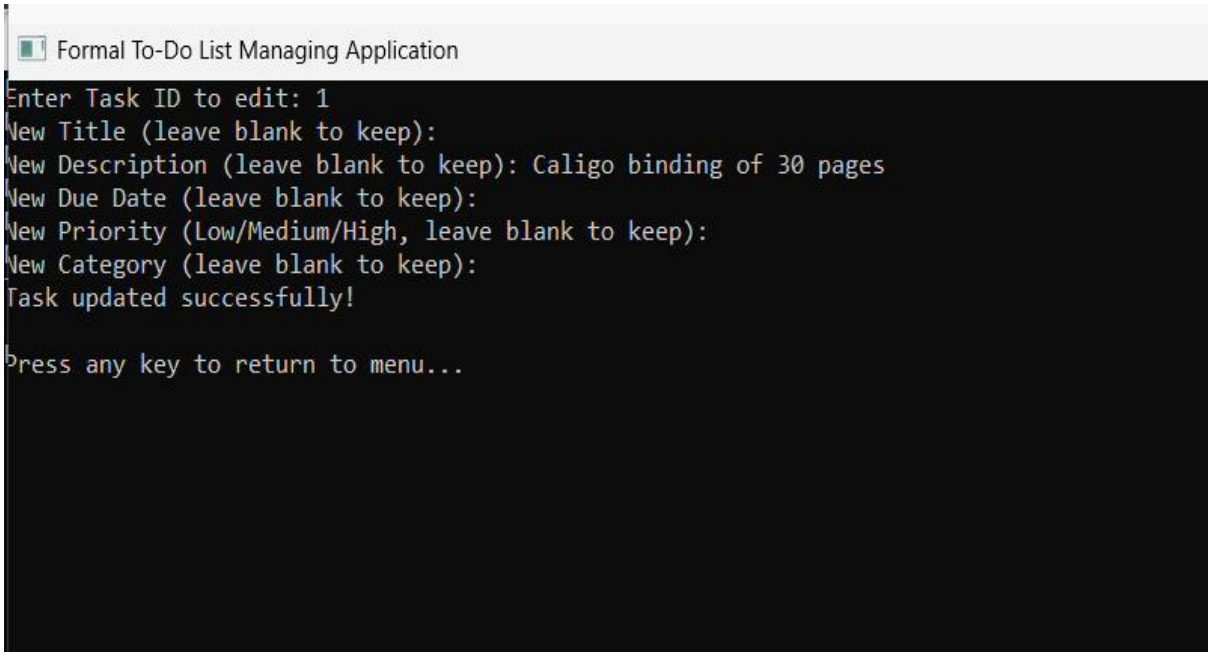


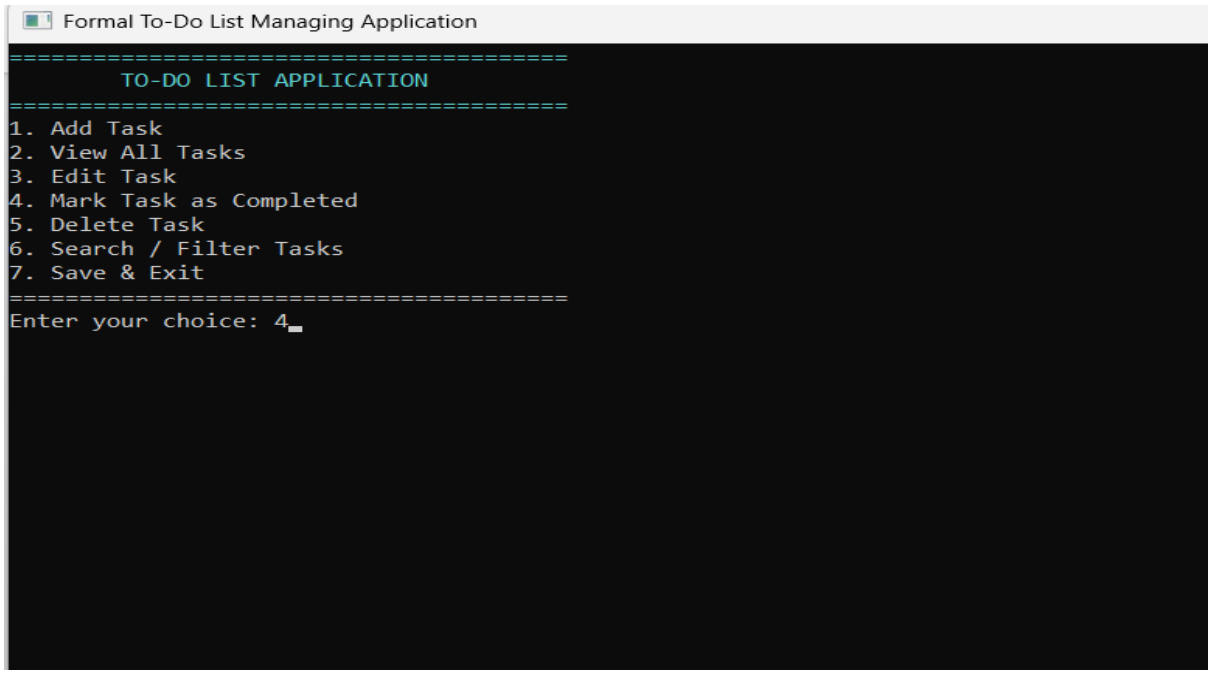

Formal To-Do List Managing Application
TO-DO LIST APPLICATION
1. Add Task
2. View All Tasks
3. Edit Task
4. Mark Task as Completed
5. Delete Task
6. Search / Filter Tasks
7. Save & Exit
Enter your choice: 2

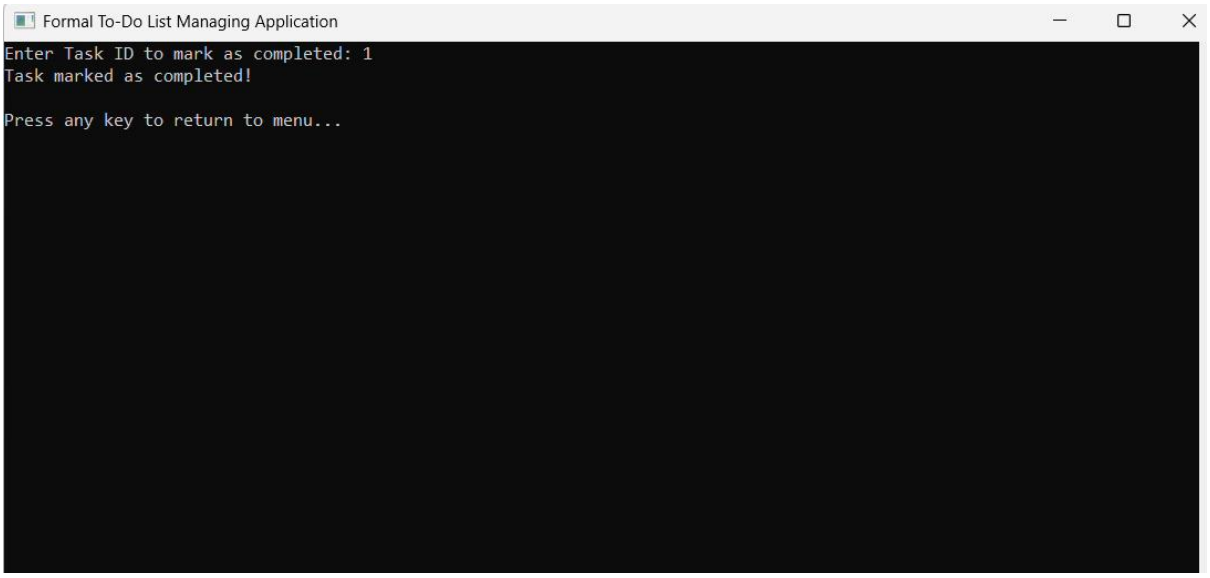

Formal To-Do List Managing Application
========= ALL TASKS =========
ID: 1 | Finish C# Micro Project | Due: 06-09-2025 | Priority: High | Category: Study | [ ] Pending
Print and Submit Caligo Document
Press any key to return to menu...

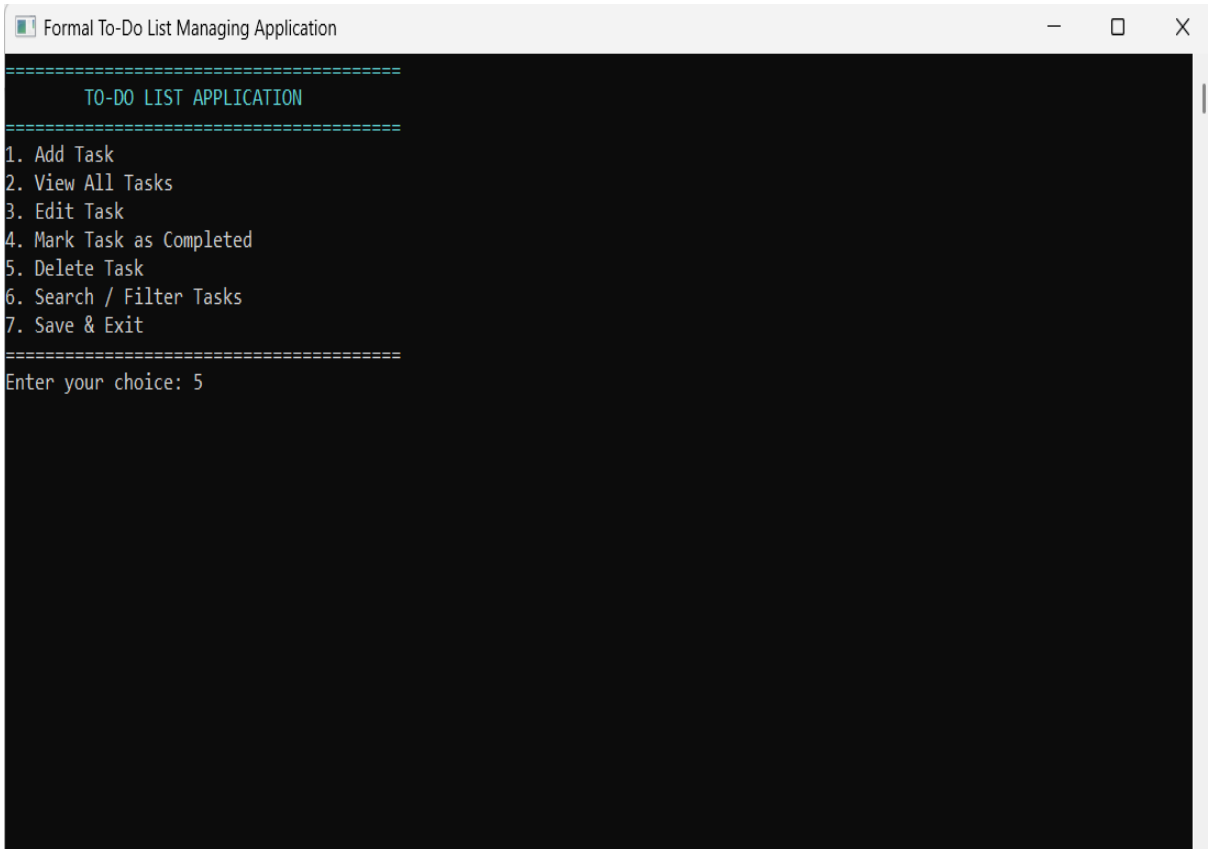

Formal To-Do List Managing Application
TO-DO LIST APPLICATION
1. Add Task
2. View All Tasks
3. Edit Task
4. Mark Task as Completed
5. Delete Task
6. Search / Filter Tasks
7. Save & Exit
Enter your choice: 3

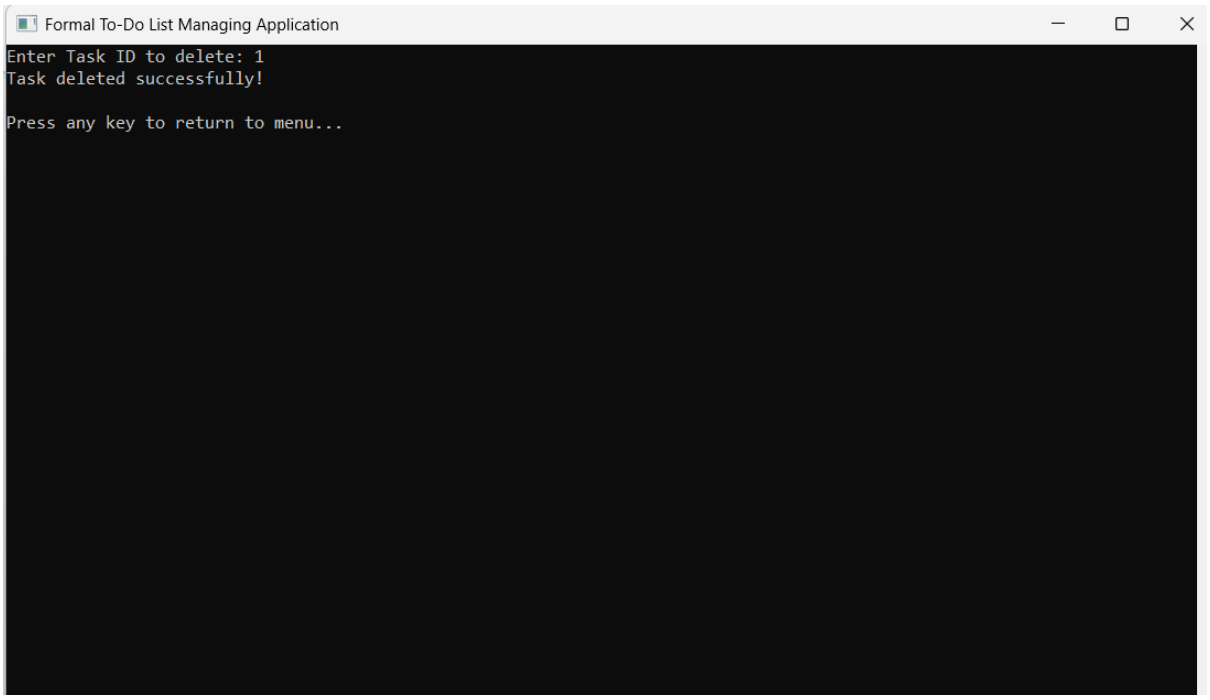

Formal To-Do List Managing Application
nter Task ID to edit: 1
lew Title (leave blank to keep):
lew Description (leave blank to keep): Caligo binding of 30 pages
lew Due Date (leave blank to keep):

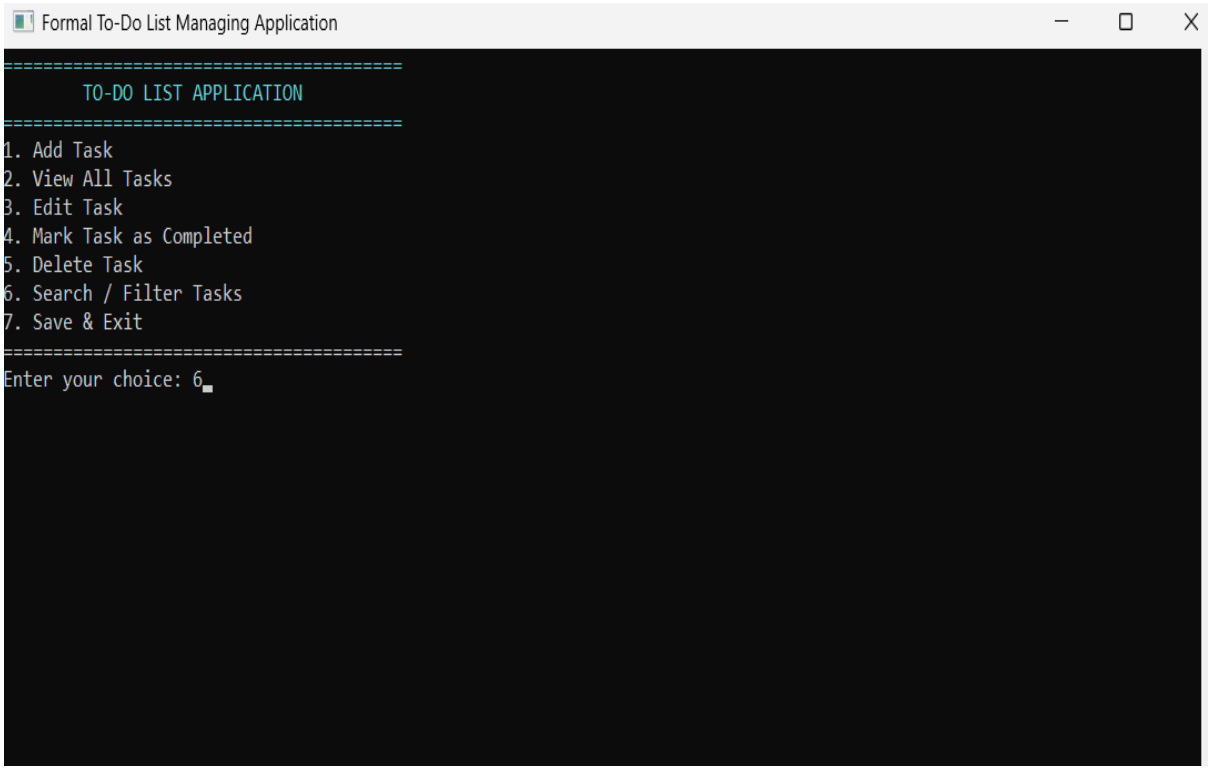

Formal To-Do List Managing Application
Enter Task ID to edit: 1
New Title (leave blank to keep):
New Description (leave blank to keep): Caligo binding of 30 pages
New Due Date (leave blank to keep):
New Priority (Low/Medium/High, leave blank to keep):
New Category (leave blank to keep):
Task updated successfully!
Press any key to return to menu...


Formal To-Do List Managing Application
TO-DO LIST APPLICATION
1. Add Task
2. View All Tasks
3. Edit Task
4. Mark Task as Completed
5. Delete Task
6. Search / Filter Tasks
7. Save & Exit
Enter your choice: 4


Formal To-Do List Managing Application
Enter Task ID to mark as completed: 1
Task marked as completed!
Press any key to return to menu...


Formal To-Do List Managing Application
TO-DO LIST APPLICATION
1. Add Task
2. View All Tasks
3. Edit Task
4. Mark Task as Completed
5. Delete Task
6. Search / Filter Tasks
7. Save & Exit
Enter your choice: 5

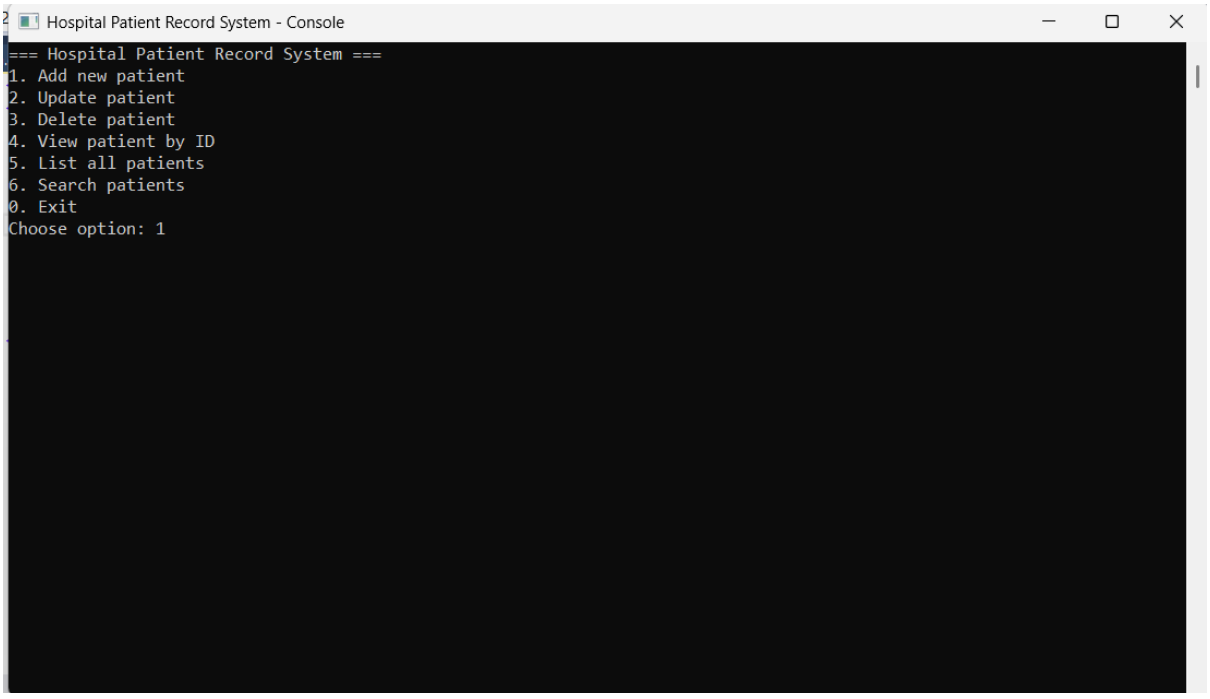

Formal To-Do List Managing Application
Enter Task ID to delete: 1
Task deleted successfully!
Press any key to return to menu...


Formal To-Do List Managing Application
TO-DO LIST APPLICATION
1. Add Task
2. View All Tasks
3. Edit Task
4. Mark Task as Completed
5. Delete Task
6. Search / Filter Tasks
7. Save & Exit
Enter your choice: 6

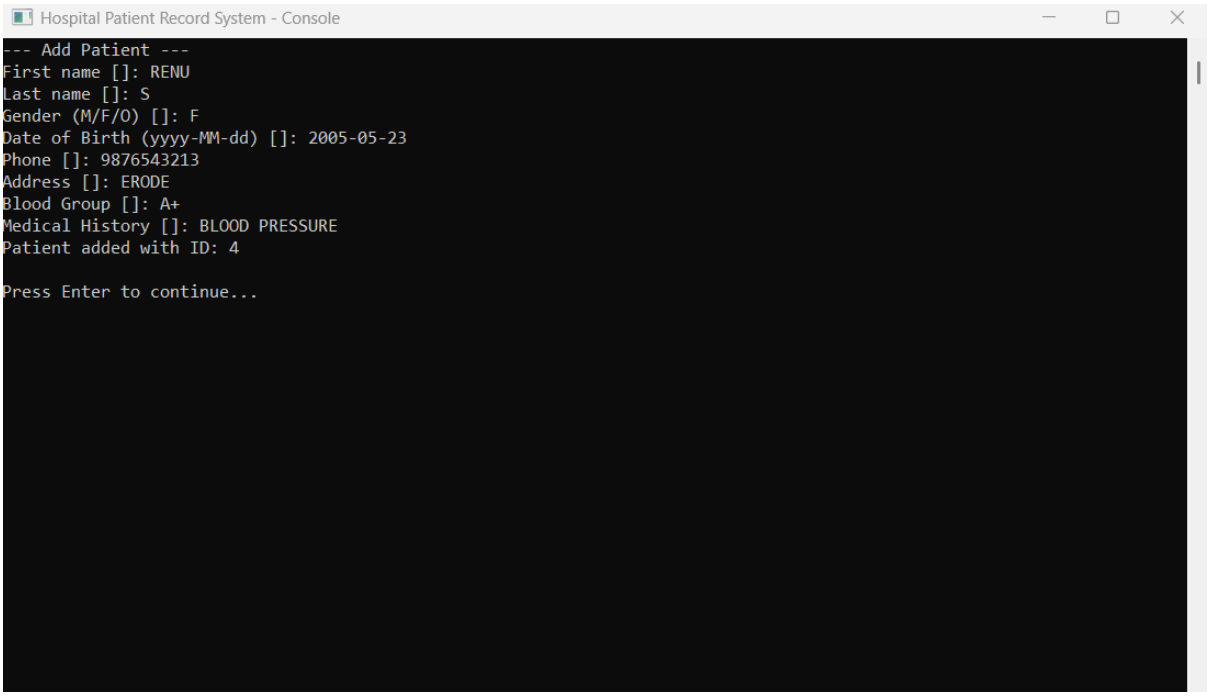
```
Search Options:
1. By Keyword
2. By Status
3. By Category
Choose option: 2
Enter status (Completed/Pending): completed
========= SEARCH RESULTS =========
ID: 1 | Finish C# Micro Project | Due: 06-09-2025 | Priority: High | Category: Study | [? Completed]
    Caligo binding of 30 pages
---------------------------------

Press any key to return to menu...
```

Hospital Application:

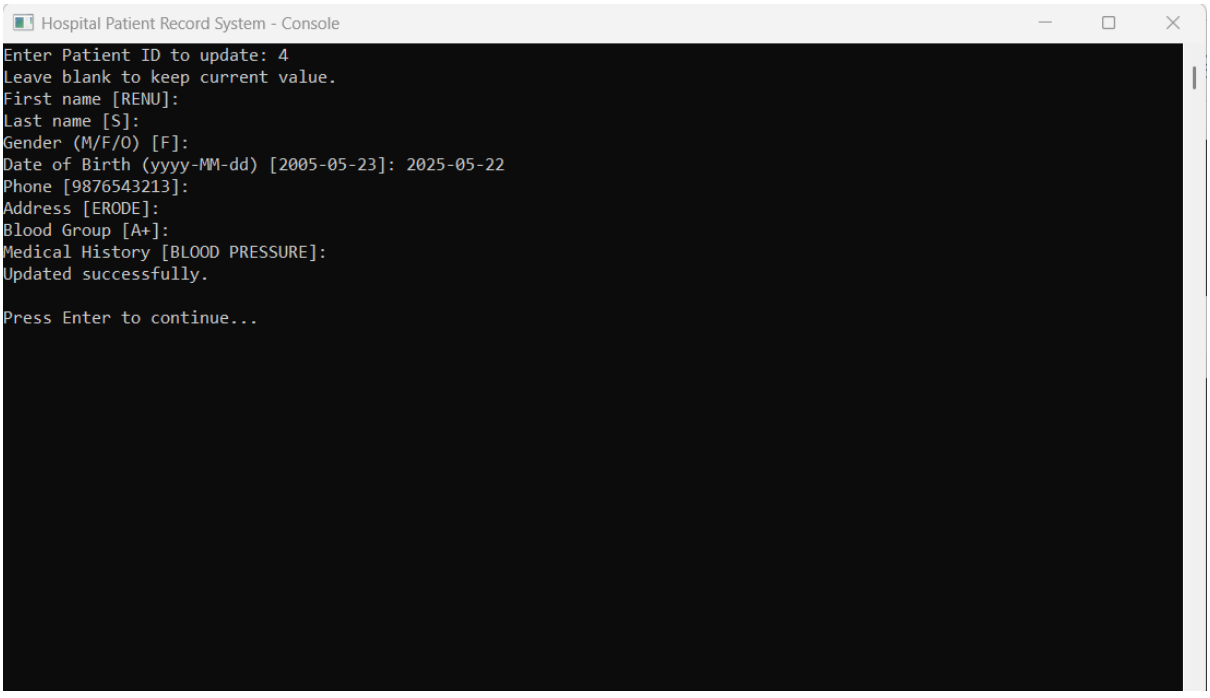
OUTPUT:



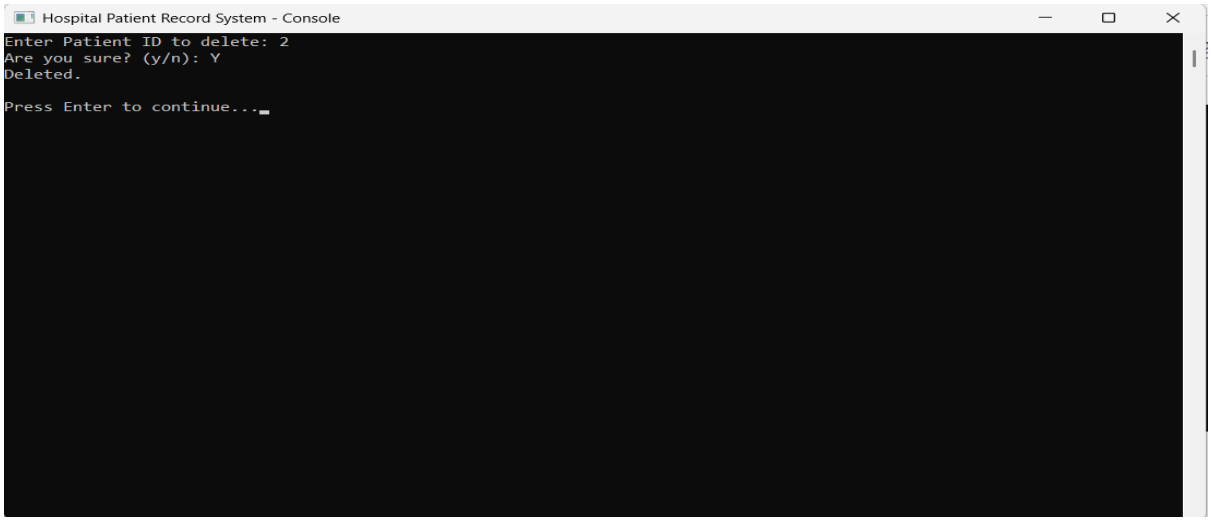
1) ADD NEW PATIENT



2) UPDATE PATIENT



3) DELETE THE PATIENT

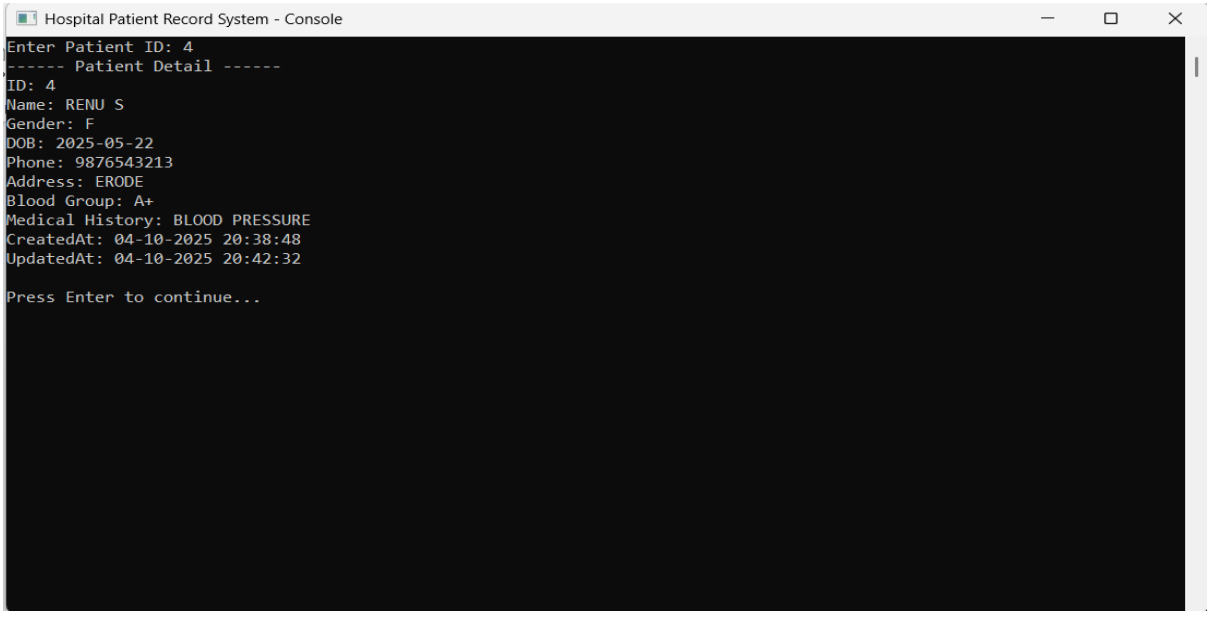


100 %

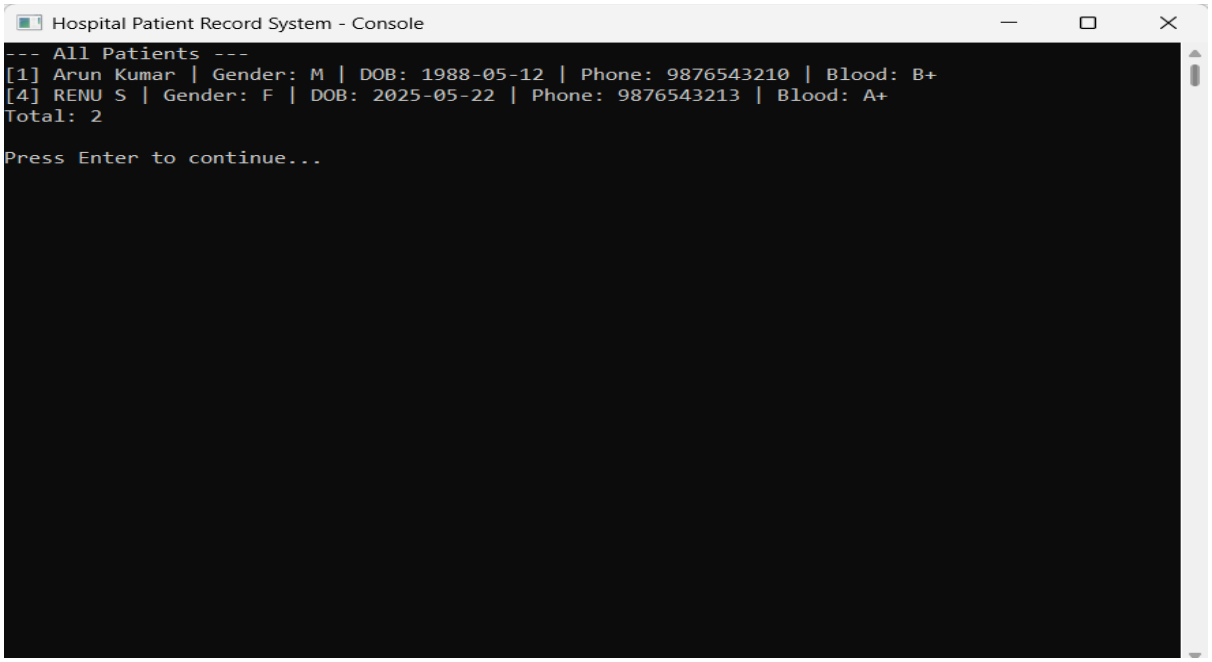
Results | Messages

	PatientId	FirstName	LastName	Gender	DateOfBirth	Phone	Address	BloodGroup	MedicalHistory	CreatedAt	UpdatedAt
1	1	Arun	Kumar	M	1988-05-12	9876543210	Chennai	B+	Hypertension	2025-10-04 18:20:01.470	NULL
2	4	RENU	S	F	2025-05-22	9876543213	ERODE	A+	BLOOD PRESSURE	2025-10-04 20:38:48.197	2025-10-04 20:42:32.243

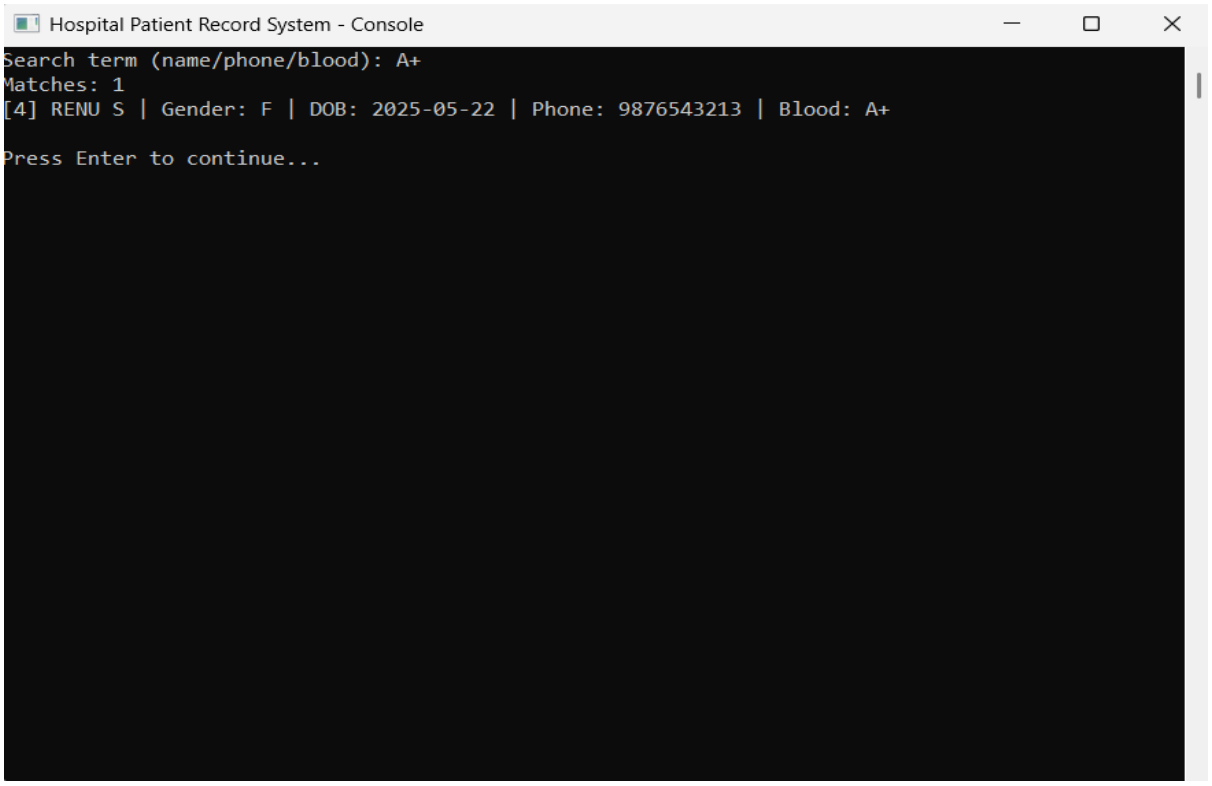
4) VIEW PATIENT BY ID



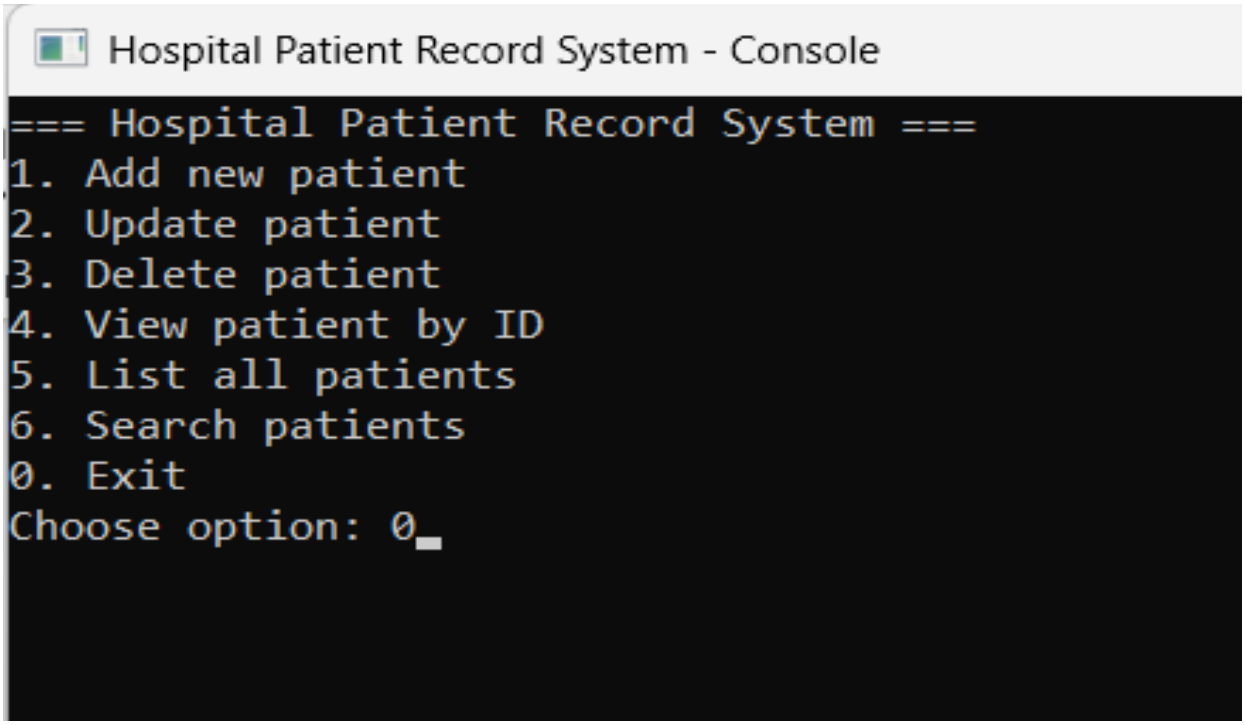
5) LIST ALL PATIENTS



6) SEARCH PATIENTS



7) EXIT



10. ADVANTAGES

- Simple and easy-to-use console interface.
- Demonstrates real-time CRUD operation handling.
- Provides both local (JSON) and database (SQL) persistence.
- Modular and extendable architecture.
- Uses OOP concepts and standard coding practices.

11. LIMITATIONS

- No graphical interface (only console-based).
- No multi-user authentication or role management.
- Limited error handling for invalid data entries.
- Lacks network or web accessibility.

12. FUTURE ENHANCEMENTS

- Upgrade to a web-based or desktop GUI (e.g., ASP.NET, WPF).
- Add user login and role-based access control.
- Implement data analytics or reporting features.
- Integrate cloud-based storage for accessibility.
- Add data export/import in CSV or Excel format.

13. CONCLUSION

Both applications successfully demonstrate how C# can be used for managing data efficiently through console-based CRUD systems. The **To-Do List App** showcases file handling and serialization, while the **Hospital Patient Record System** illustrates database connectivity and record management using ADO.NET. These projects provide a strong foundation for understanding backend development, OOP concepts, and real-world application design.