Web Engineering SEF22M

Assignment 1

Deadline:Tuesday 8th October 2024 (11:59pm)

Hospital Management System in .NET (C#)

Important Instructions:

- Ensure proper indentation and maintain clean, tidy code with classes distributed across files.
- Verify that all functions are fully operational.
- Code must be free of syntax errors.
- Handle all necessary exceptions and validation checks meticulously in each function; failure to do so will result in a zero grade for that function.
- Emphasize creating custom logic rather than relying on built-in functions.

Objective:

Create a console-based **Hospital Management System** using .NET (C#) that can manage hospital operations, including patients, doctors, and appointments. The system will first use file-based data storage in JSON format, then transition to a database-driven solution using ADO.NET. The assignment will cover advanced topics like serialization, deserialization, object initializer syntax, constructors, and data types storage in stack and heap, along with database management using ADO.NET.

Case Study

Scenario:

The **City Hospital** is undergoing digital transformation. Their manual system for managing patients, doctors, and appointments is outdated, leading to inefficiencies. You have been tasked with building a management system that will first store data using JSON files and then transition to a database solution using ADO.NET.

Project Type Explanation

Project Type: This project consists of two separate components:

1. **Data Access Layer (DAL):**

    - **Project Type:** Class Library
    - **Purpose:** To manage all data-related operations. This layer is responsible for handling CRUD (Create, Read, Update, Delete)

operations on the library's data. It abstracts the complexities of data storage and retrieval from the rest of the application, promoting a modular and maintainable code structure.

- **Data Handling:** Use ADO.NET to create a database, tables, and perform all CRUD operations.

2. **Presentation Layer (Console Application):**

    - **Project Type:** Console Application
    - **Purpose:** To provide a user interface for interacting with the Hospital management system. This layer presents a menu-driven interface that allows users to perform various operations like adding, deletion, updation, booking, searching etc,
    - **Interaction:** Acts as the mediator between the user and the DAL, handling user inputs and displaying the results of data operations.

Data Access Layer (DAL)

Project Name: HospitalDAL

Classes and Properties

1. **Patient:**
    - int PatientId
    - string Name
    - string Email
    - string Disease
2. **Doctor:**
    - int DoctorId
    - string Name
    - string Specialization
3. **Appointment:**
    - int AppointmentId
    - int PatientId
    - int DoctorId
    - DateTime AppointmentDate

Function Prototypes:

- **Insert Records into the Database**:

    - void InsertPatient(Patient patient);
    - void InsertDoctor(Doctor doctor);
    - void InsertAppointment(Appointment appointment);

- **Read Records from the Database**:

- List<Patient> GetAllPatientsFromDatabase();
- List<Doctor> GetAllDoctorsFromDatabase();
- List<Appointment> GetAllAppointmentsFromDatabase();

- **Update Records in the Database**:

  - void UpdatePatientInDatabase(Patient patient);
  - void UpdateDoctorInDatabase(Doctor doctor);
  - void UpdateAppointmentInDatabase(Appointment appointment);

- **Delete Records from the Database**:

  - void DeletePatientFromDatabase(int patientId);
  - void DeleteDoctorFromDatabase(int doctorId);
  - void DeleteAppointmentFromDatabase(int appointmentId);

- **Search Records in the Database**:

  - List<Patient> SearchPatientsInDatabase(string name);
  - List<Doctor> SearchDoctorsInDatabase(string specialization);
  - List<Appointment> SearchAppointmentsInDatabase(int doctorId, int patientId);

Additional Tips:

Save Deleted Record in a History File:

- Before the actual deletion takes place, the record being deleted should be saved to a history file.
- **JSON File (for history storage):**
  - Store the deleted record in a separate file (e.g., DeletedPatients.json, DeletedDoctors.json, or DeletedAppointments.json).
  - If the history file doesn't exist, create it; otherwise, append the deleted record.
  - Serialize the deleted record and append it to the history file so that you have a log of all deleted records for future reference.
  - Ensure each record saved in the history file contains a **timestamp** and additional metadata (such as the date of deletion).

Presentation Layer (Console Program)

Project Name: HospitalManagementConsoleApp

Menu Options:

1. Add a new patient
2. Update a patient
3. Delete a patient (and save deleted record to history)
4. Search for patients by name
5. View all patients

6. Add a new doctor
7. Update a doctor
8. Delete a doctor (and save deleted record to history)
9. Search for doctors by specialization
10. View all doctors
11. Book an appointment
12. View all appointments
13. Search appointments by doctor or patient
14. Cancel an appointment (and save deleted appointment to history)
15. View history of deleted records (patients, doctors, or appointments)
16. Exit the application

Validation and Error Handling:

Input Validation

- Ensure that user input is valid before processing or storing data.
  - **Required Fields**: Ensure that all mandatory fields (e.g., name, email, etc.) are filled in.
  - **Email Format Validation**: Ensure that the email address is valid.
  - **ID Uniqueness**: PatientId, DoctorId, and AppointmentId should be unique.
  - **Appointment Date Validation**: Ensure that appointment dates are in the future.
  - **Prevent Duplicate registration**
  - Check appointment slots before assigning assignment (datetime do not match with any previously assigned DateTime of appointment)

Submission Requirements:

Submission will be through **GitHub Classroom**; the link will be shared later.

Project Structure:

```
HospitalManagementSystem/
|
├── HospitalDAL/                    // Data Access Layer (DAL)
|   ├── HospitalDAL.csproj          // Project file for the Data Access Layer
|   ├── Patient.cs                  // Patient entity class
|   ├── Doctor.cs                   // Doctor entity class
|   ├── Appointment.cs              // Appointment entity class
|   ├── DataAccess.cs               // Handles CRUD operations for patients, doctors, ap
|   ├── History.cs                  // Handles saving and retrieving deleted records (pa
|
└── HospitalConsoleApp/             // Console Application (Presentation Layer)
    ├── HospitalConsoleApp.csproj   // Project file for the Console Application
    ├── Program.cs                  // Main entry point for the console application
    ├── Menu.cs                     // Menu system for user interaction
```

You need to follow the specified structure for your project. Please note the following important points:

1. **Single Solution with Multiple Projects**:
    - You will have **one solution** (e.g., **HospitalManagementSystem**), and within this solution, you should create **two separate projects**:
        - One for the **Data Access Layer (DAL)**.
        - One for the **Console Application (Presentation Layer)**.
2. **DAL Must Be a Class Library**:
    - The **DAL (HospitalDAL)** must be implemented as a **Class Library**. This means it should not contain any UI logic but only focus on data operations such as creating, reading, updating, and deleting records (CRUD operations) using ADO.NET and handling JSON files for history.
    - The DAL will be referenced by the Console Application to interact with the database and other data sources.