****

**Hospital Management system**

**Object Oriented Programming**



**January 24, 2025**

**Brithway Julius**

**Student No: 20029570**

|  |  |
| --- | --- |
| **NO** | **TABLE OF CONTENT** |
| **1** | [**Project Overview**](#_Project_Overview) |
| **2** | [**UML Class Diagram**](#_UML_CLASS_DIAGRAM) |
| **3** | [**Code**](#_Code) |
| **4** | [**Screenshots**](#_Screenshot’s) |
| **5** | [**Reference**](#_Reference) |

# **Project Overview**

**Objective:**

“The rapid growth in Information & Communication Technology (ICT), and the power of Internet has strongly impacted the business and service delivery models of today ‘s global environment. E-Hospital Management Systems provide the benefits of streamlined operations, enhanced administration & control, superior patient care, strict cost control and improved profitability.

Hospital Information systems are in high demand to handle increasing population needs and aids the practicing doctors and hospital service and support staff with timely service and precision. There are varied metrics available to assess the performance of services like hospital industry, and the successful implementation and usage of Hospital information system forms a crucial role. Hospital information systems are available in the software market which in most cases needs to be customized and in some cases HIS needs to be developed as a customized software based on specific hospital requirements (user requirements).”

The project aims to create an efficient hospital management system which manages hospital staff member and patient data effectively. The system enables medical staff to view existing data as well as create and operate doctor records along with nurse records and patient records and staff records of receptionists and technicians. Users can execute queries against their data to retrieve specific informational insights including targeted staff filters and patient age group listings.

**Scope:**

This system is designed to streamline hospital operations by providing the following key features:

* **Add Records:**
  + The system enables new registrations for doctors along with nurses as well as patients and receptionists and technicians so users can add their details properly.
  + The structured data entry format is achieved through separate forms for each category.
* **View Records:**
  + The application shows a structured grid view containing doctor, nurse, patient and receptionist and technician listings.
  + Perform custom queries to filter data, such as:
    - List of male nurses.
    - Patients above the age of 25.
    - Any other two queries of the user’s choice.
* **Database Management:**
  + All system data is safely stored in a relational database structure.
  + The designed system includes database functionalities that handle the relationships between doctors’ nurses’ patients’ receptionists and technicians’ entities.
* **Object-Oriented Design:**
  + Strong object-oriented programming principles form the foundation which guides the project design.
  + The system includes six real-world classes representing Person, Doctor, Nurse, Patient, Receptionist and Technician respectively along with their associated attributes and methods.
  + The design follows modular and extensible patterns through the implementation of inheritance together with encapsulation principles.
* **User Interface:**
* The system presents a user interface which enables users to perform operations through forms that help record entries and display results.
* Users can navigate through the interface by using dropdown selections and buttons which are quick to understand.
* **Custom Queries:**
* Users can run specific queries to gain insights, such as:
* All nursing staff members who have a male gender.
* The interface shows patients who reach a defined age threshold.
* The system allows users to run two distinct query functions which extract data using specified criteria.
* **Scalability:**
* The system demonstrates scalability through its design which enables new staff categories addition and functional expansion while requiring minimal system modifications.

**Technologies Used:**

* **Programming Language:** C# (Windows Forms for GUI development).

“The C# project started more than 12 years ago, in December 1998, with the goal to

create a simple, modern, object-oriented, and type-safe programming language for

the new and yet-to-be-named .NET platform. Since then, C# has come a long way.

The language is now in use by more than a million programmers and has been

released in four versions, each with several major new features added.

C# (pronounced "See Sharp") is a simple, modern, object-oriented, and type-safe

programming language. C# has its roots in the C family of languages and will be

immediately familiar to C, C++, and Java programmers. C# is standardized by ECMA

International as the ECMA-334 standard and by ISO/IEC as the ISO/IEC 23270

standard. Microsoft's C# compiler for the .NET Framework is a conforming

implementation of both standards.”

* **Database:** SQL Server functions as the main system for both data storage and management.

“Conceived in the mid-1970s as a database language for the then new relational

model, SQL was first standardized in 1986 and was modestly enhanced in 1989. In

1992, work was completed on a significantly revised version of the SQL standard.

SQL products and implementations are now so widespread that it would be

extremely difficult for any active programmer or applications developer to do his

or her work without familiarity with the current SQL standard. Even today's

personal computer (PC) database products are heavily influenced by SQL as more

and more of those products are integrated with software environments from

midrange and mainframe computers.”

* **Tools for UML Design:** Draw.io functions to develop class diagrams.

“Software engineering is a discipline utilizing Unified Modelling Language (UML) diagrams, which are accepted as a standard to depict object-oriented design models. UML diagrams make it easier to identify the requirements and scopes of systems and applications by providing visual models. In this manner, this study aims to systematically review the literature on UML diagram utilization in software engineering research. A comprehensive review was conducted over the last two decades, spanning from 2000 to 2019. Among several papers, 128 were selected and examined. The main findings showed that UML diagrams were mostly used for the purpose of design and modelling, and class diagrams were the most used ones.

The Unified Modelling Language (UML) is also used to develop a system in software engineering, which is a visual language to define and document a system. The requirements in scenarios that express how users use a system are shown with the UML. The constraints of a system are also shown with the UML. Hence, many researchers who work as software engineers publish papers about how UML diagrams are utilized to develop a system and contribute to the practice to advance the software engineering discipline. In our study, SLR is used to understand which UML diagrams are popular, why they are used, and which application areas are the most popular.”

**Features Summary:**

* A system provides centralized control over staff records alongside patient records.
* The system employs object-oriented principles to achieve extensibility and maintain its operational capabilities.
* The platform provides flexible search functions that allow users to retrieve insights from existing database records.
* Intuitive user interface for smooth interaction.

## **UML CLASS DIAGRAM**

**Person Class**

* **Description:**

The Person class acts as base class that underlies every entity throughout the system. The Person class acts as the base foundation for all system entities to share their common attributes with inherited methods.

* **Attributes:**
* Name: The name of the person (string).
* Address: The address of the person (string).
* County: The County of the Person (string).
* Age: The age of the person (integer).
* Phone: The Phone No of the Person (string).
* Email: The Email of the Person (string).
* **Methods:**
* GetDetails ()
* **Relationships:**

From its foundation The Person class establishes foundations used by sub-classes including Staff and Patient.

**Staff Class**

* **Description:**

The Staff class represents any Staff Member in the Hospital. The Staff Class inherits from the Person Class, and it includes roles like Doctors, Nurse, Receptionist, Technician.

* **Attributes:**
* StaffCode: The StaffCode of the Staff (string).
* Department: The Department of the Staff (string).
* **Methods:**
* AssignDepartment ()
* GetStaffDetails ()
* **Relationships:**

The Staff class establishes foundations used by sub-classes including Doctor, Nurse, Receptionist, Technician.

**Doctor Class**

* **Description:**

The role of hospital doctor is represented by this class. Inherits from the Staff class.

* **Attributes:**
* Pay: The Pay of the Doctor (decimal).
* **Methods:**
* AssignPay ()
* AddDoctor ()
* GetDoctorDetails ()
* **Relationships:**

Inherits from Staff class.

**Nurse Class**

* **Description:**

The role of hospital Nurse is represented by this class. Inherits from the Staff class.

* **Attributes:**
* Hours: The Hours of the Nurse Working (string).
* Gender: The Gender of the Nurse (string).
* **Methods:**
* AssignHours ()
* AssignGender ()
* AddNurse ()
* GetNurseDetails ()
* **Relationships:**

Inherits from Staff class.

**Patient Class**

* **Description:**

The role of hospital Patient by this class. Inherits from the Person class.

* **Attributes:**
* PatientNumber: The Patient No of the Patient (string).
* **Methods:**
* AssignPatientNumber ()
* AddPatient ()
* GetPatientDetails ()
* **Relationships:**

Inherits from Person class.

**Receptionist Class**

* **Description:**

The role of hospital Receptionist is represented by this class. Inherits from the Staff class.

* **Attributes:**
* Shift: The Shift of the Receptionist (string).
* **Methods:**
* AddRecptionist ()
* ScheduleAppointments ()
* RegisterPatients ()
* HandlePhoneCalls ()
* GetReceptionistDetails ()
* **Relationships:**

Inherits from Staff class.

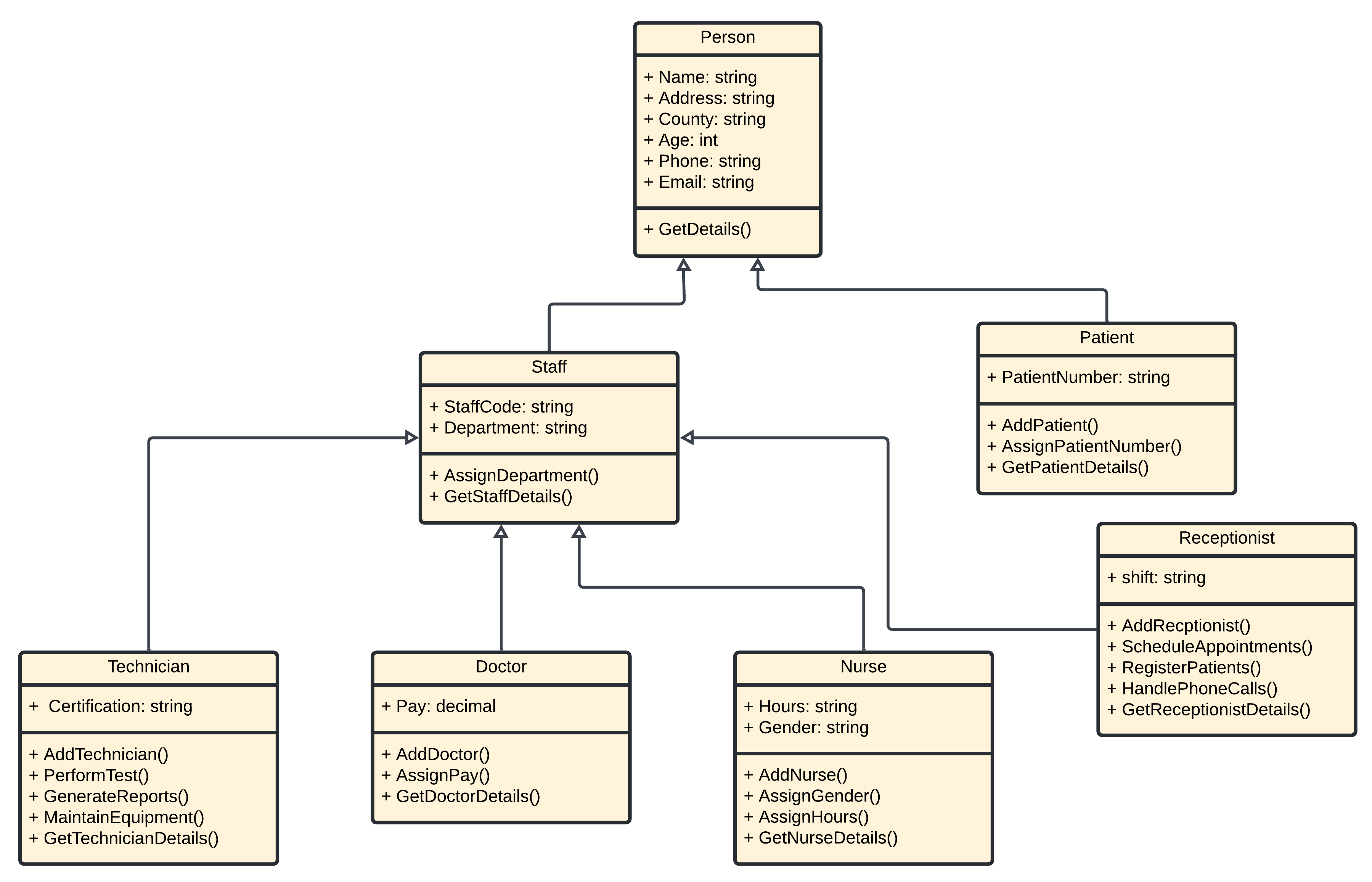
**Technician Class**

* **Description:**

The role of hospital Technician is represented by this class. Inherits from the Staff class.

* **Attributes:**
* Certification: The Certification of the Technician (string).
* **Methods:**
* AddTechnician ()
* PerformTest ()
* GenerateReports ()
* MaintainEquipment ()
* GetTechnicianDetails ()
* **Relationships:**

Inherits from Staff class.



A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.A computer screen shot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

### **Code**

**Main form Code**

* **Form1.cs**

namespace Hospital\_Managament\_System

{

public partial class Form1 : Form

{

// Instantiate helper objects for hash generation and database operations.

Hashcode hs = new Hashcode();

DAO dao = new DAO();

// Constructor for the form. Initializes the UI components.

public Form1()

{

InitializeComponent();

}

// Event handler for the Login button click event.

private void btnLogin\_Click(object sender, EventArgs e)

{

// Retrieve the username and hash the password input using the Passhash method.

string Username = tbxUn.Text;

string Password = hs.Passhash(tbxPd.Text);

SqlDataReader dr = null; // To hold the result of the database query.

// Create a SQL command object using the DAO's open connection.

SqlCommand cmd = dao.OpenCon().CreateCommand();

// Specify the stored procedure for login.

cmd.CommandText = "usplogin";

cmd.CommandType = CommandType.StoredProcedure;

// Add parameters to the SQL command for the username and password.

cmd.Parameters.AddWithValue("@Un", Username);

cmd.Parameters.AddWithValue("@Pd", Password);

// Execute the command and retrieve results.

dr = cmd.ExecuteReader();

// Check if the reader contains any data (indicating successful login).

if (dr.Read())

{

// Inform the user of a successful login.

MessageBox.Show("You Are Logged in");

// Clear the input fields.

tbxUn.Clear();

tbxPd.Clear();

// Display the logged-in user's name.

Name = Username;

lblDisplay.Text = "Welcome " + Name;

// Disable login-related UI elements after successful login.

lblPd.Enabled = false;

lblUn.Enabled = false;

tbxPd.Enabled = false;

tbxUn.Enabled = false;

btnRegister.Enabled = false;

btnLogin.Enabled = false;

}

else

{

// Inform the user of an invalid login attempt.

MessageBox.Show("Invalid");

}

// Close the database connection after the operation.

dao.CloseCon();

}

// Event handler for the Register button click event.

private void btnRegister\_Click(object sender, EventArgs e)

{

// Open the Register form as a dialog.

Register register = new Register();

register.ShowDialog();

}

// Event handler for the "Add Doctor" menu item click event

private void addDoctorToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Create a new instance of the Doctor form.

Doctor doctor = new Doctor();

// Display the Doctor form as a modal dialog.

doctor.ShowDialog();

}

// Event handler for the "Add Nurse" menu item click event.

private void addNurseToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Create a new instance of the Nurse form.

Nurse nurse = new Nurse();

// Display the Nurse form as a modal dialog.

nurse.ShowDialog();

}

// Event handler for the "Add Patient" menu item click event.

private void addPatientToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Create a new instance of the Patient form.

Patient patient = new Patient();

// Display the Patient form as a modal dialog.

patient.ShowDialog();

}

// Event handler for the "Add Receptionist" menu item click event.

private void addReceptionistToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Create a new instance of the Receptionist form.

Receptionist receptionist = new Receptionist();

// Display the Technician form as a modal dialog.

receptionist.ShowDialog();

}

// Event handler for the "View Records" menu item click event.

private void addTechnicianToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Create a new instance of the View form.

Technician technician = new Technician();

// Display the View form as a modal dialog to show records.

technician.ShowDialog();

}

private void viewRecordsToolStripMenuItem\_Click(object sender, EventArgs e)

{

View view = new View();

view.ShowDialog();

}

// Event handler for the "Exit" menu item click event.

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Close the current form to exit the application.

this.Close();

}

}

}

* **Register.cs**

namespace Hospital\_Managament\_System

{

public partial class Register : Form

{

// Constructor for the Register form. Initializes the UI components.

public Register()

{

InitializeComponent();

}

// Create instances of helper classes for hashing passwords and database operations.

Hashcode hs = new Hashcode();

DAO dao = new DAO();

// Event handler for the Register button click event.

private void btnRegister\_Click(object sender, EventArgs e)

{

// Retrieve the full name, username, and hashed password from the input fields.

string FullName = txtFn.Text;

string UserName = txtUn.Text;

string Password = hs.Passhash(txtPd.Text);

// Set up the SQL command to interact with the database using a stored procedure.

SqlCommand cmd = dao.OpenCon().CreateCommand();

cmd.CommandText = "uspRegister"; // Name of the stored procedure for user registration.

cmd.CommandType = CommandType.StoredProcedure;

// Add parameters to the command for the stored procedure.

cmd.Parameters.AddWithValue("@Fn", FullName); // Full name parameter.

cmd.Parameters.AddWithValue("@Un", UserName); // Username parameter.

cmd.Parameters.AddWithValue("@Pd", Password); // Password parameter (hashed).

// Execute the SQL command to insert the new user into the database.

cmd.ExecuteNonQuery();

// Close the database connection after completing the operation.

dao.CloseCon();

// Clear the input fields to reset the form.

txtFn.Clear();

txtUn.Clear();

txtPd.Clear();

// Close the registration form after successful registration.

this.Close();

}

}

}

* **Doctor.cs**

namespace Hospital\_Managament\_System

{

public partial class Doctor : Form

{

// Constructor for initializing the form

public Doctor()

{

InitializeComponent(); // Initializes the form components (e.g., buttons, textboxes, etc.)

}

// Event handler for the "Exit" button click event

private void btnExit\_Click(object sender, EventArgs e)

{

// Create a new instance of Form1 and display it

Form1 form1 = new Form1();

form1.ShowDialog(); // Show Form1 as a dialog, meaning the current form is paused until it's closed

}

// Event handler for the "Submit" button click event

private void btnSubmit\_Click(object sender, EventArgs e)

{

// Retrieve values entered by the user in the form's input fields

string ne = tbxName.Text; // Get the doctor's name from the textbox

string ad = tbxAddress.Text; // Get the doctor's address from the textbox

string cy = cbxCounty.Text; // Get the selected county from the combobox

int ag = int.Parse(tbxAge.Text); // Convert the age input from textbox to an integer

string ph = tbxPhone.Text; // Get the doctor's phone number from the textbox

string em = tbxEmail.Text; // Get the doctor's email address from the textbox

string se = tbxStaffCode.Text; // Get the doctor's staff code from the textbox

string dt = cbxDepartment.Text; // Get the selected department from the combobox

decimal py = decimal.Parse(tbxPay.Text); // Convert the pay from textbox to a decimal

// Create an instance of the AddDoctor class and pass the entered details

AddDoctor addDoctor = new AddDoctor(ne, ad, cy, ag, ph, em, se, dt, py);

// Call the method to add the new doctor to the database

addDoctor.AddNewDoctor();

// Clear all input fields after the doctor information is submitted

tbxName.Clear();

tbxAddress.Clear();

tbxPhone.Clear();

tbxEmail.Clear();

tbxPay.Clear();

tbxAge.Clear();

tbxStaffCode.Clear();

}

}

}

* **Nurse.cs**

namespace Hospital\_Managament\_System

{

public partial class Nurse : Form

{

public Nurse()

{

InitializeComponent();

}

// Event handler for the "Exit" button click event

private void btnExit\_Click(object sender, EventArgs e)

{

// Event handler for the "Exit" button click event

Form1 form1 = new Form1();

form1.ShowDialog(); // Show Form1 as a dialog (pauses the current form until it's closed)

}

// Event handler for the "Submit" button click event

private void btnSubmit\_Click(object sender, EventArgs e)

{

// Retrieve values entered by the user in the form's input fields

string ne = tbxName.Text;

string ad = tbxAddress.Text;

string cy = cbxCounty.Text;

int ag = int.Parse(tbxAge.Text);

string ph = tbxPhone.Text;

string em = tbxEmail.Text;

string se = tbxStaffCode.Text;

string dt = cbxDepartment.Text;

string ge = "Male";

if (rdbFemale.Checked)

ge = "Female";

string hr = tbxHours.Text;

// Create an instance of the AddNurse class, passing all the entered details

AddNurse addNurse = new AddNurse(ne, ad, cy, ag, ph, em, se, dt, hr, ge);

// Call the method to add the new nurse to the data store

addNurse.AddNewNurse();

// Clear all the input fields after the nurse's information is submitted

tbxName.Clear();

tbxAddress.Clear();

tbxPhone.Clear();

tbxEmail.Clear();

tbxAge.Clear();

tbxHours.Clear();

tbxPhone.Clear();

tbxStaffCode.Clear();

}

}

}

* **Patient.cs**

namespace Hospital\_Managament\_System

{

public partial class Patient : Form

{

public Patient()

{

InitializeComponent(); // Initialize the components of the form

}

private void btnExit\_Click(object sender, EventArgs e)

{

Form1 form1 = new Form1(); // Create an instance of Form1

form1.ShowDialog(); // Display Form1 as a dialog box

}

private void btnSubmit\_Click(object sender, EventArgs e)

{

// Collect user input from various textboxes

string ne = tbxName.Text; // Patient's name

string ad = tbxAddress.Text; // Patient's address

string cy = cbxCounty.Text; // Patient's county

int ag = int.Parse(tbxAge.Text); // Patient's age

string ph = tbxPhone.Text; // Patient's phone number

string em = tbxEmail.Text; // Patient's email address

string pn = tbxPatientNumber.Text; // Patient's unique identifier

// Create an instance of AddPatient and add a new patient record

AddPatient addPatient = new AddPatient(ne, ad, cy, ag, ph, em, pn);

addPatient.AddNewPatient(); // Call the method to save the patient data

// Clear all input fields after submission

tbxName.Clear();

tbxAddress.Clear();

tbxPhone.Clear();

tbxEmail.Clear();

tbxAge.Clear();

tbxPatientNumber.Clear();

}

}

}

* **Receptionist.cs**

namespace Hospital\_Managament\_System

{

public partial class Receptionist : Form

{

public Receptionist()

{

InitializeComponent();

}

private void btnSubmit\_Click(object sender, EventArgs e)

{

// Retrieve values entered by the user in the form's input fields

string ne = tbxName.Text; // Get the doctor's name from the textbox

string ad = tbxAddress.Text; // Get the doctor's address from the textbox

string cy = cbxCounty.Text; // Get the selected county from the combobox

int ag = int.Parse(tbxAge.Text); // Convert the age input from textbox to an integer

string ph = tbxPhone.Text; // Get the doctor's phone number from the textbox

string em = tbxEmail.Text; // Get the doctor's email address from the textbox

string se = tbxStaffCode.Text; // Get the doctor's staff code from the textbox

string dt = cbxDepartment.Text; // Get the selected department from the combobox

string st = tbxShift.Text;

// Create an instance of AddReceptionist and add a new receptionist record

AddReceptionist addReceptionist = new AddReceptionist(ne, ad, cy, ag, ph, em, se, dt, st);

addReceptionist.AddNewReceptionist(); // Call the method to save the Receptionist data

// Clear all input fields after submission

tbxName.Clear();

tbxAddress.Clear();

tbxPhone.Clear();

tbxEmail.Clear();

tbxAge.Clear();

tbxStaffCode.Clear();

tbxShift.Clear();

}

private void btnExit\_Click(object sender, EventArgs e)

{

Form1 form1 = new Form1();

form1.ShowDialog();

}

}

}

* **Technician.cs**

namespace Hospital\_Managament\_System

{

public partial class Technician : Form

{

public Technician()

{

InitializeComponent();

}

private void btnSubmit\_Click(object sender, EventArgs e)

{

// Retrieve values entered by the user in the form's input fields

string ne = tbxName.Text; // Get the doctor's name from the textbox

string ad = tbxAddress.Text; // Get the doctor's address from the textbox

string cy = cbxCounty.Text; // Get the selected county from the combobox

int ag = int.Parse(tbxAge.Text); // Convert the age input from textbox to an integer

string ph = tbxPhone.Text; // Get the doctor's phone number from the textbox

string em = tbxEmail.Text; // Get the doctor's email address from the textbox

string se = tbxStaffCode.Text; // Get the doctor's staff code from the textbox

string dt = cbxDepartment.Text; // Get the selected department from the combobox

string ct = tbxCert.Text;

// Create an instance of AddReceptionist and add a new receptionist record

AddTechnician addTechnician = new AddTechnician(ne, ad, cy, ag, ph, em, se, dt, ct);

addTechnician.AddNewTechnician(); // Call the method to save the Receptionist data

// Clear all input fields after submission

tbxName.Clear();

tbxAddress.Clear();

tbxPhone.Clear();

tbxEmail.Clear();

tbxAge.Clear();

tbxStaffCode.Clear();

tbxCert.Clear();

}

private void btnExit\_Click(object sender, EventArgs e)

{

Form1 frm = new Form1();

frm.ShowDialog();

}

}

}

**Database Connection and View Code**

* **App.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<connectionStrings>

<!-- Connection for SQl -->

<add name="DBCon" providerName="System.Data.SqlClient" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\james\Desktop\OOP Assingments\Hospital Managament System\Hospital Managament System\Hospital.mdf;Integrated Security=True"/>

</connectionStrings>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />

</startup>

</configuration>

* **DAO.cs**

namespace Hospital\_Managament\_System

{

internal class DAO // Data Access Object

{

// Declare a SqlConnection object to interact with the database.

SqlConnection con;

// Constructor to initialize the connection object.

// Retrieves the connection string from the configuration settings.

public DAO()

{

// Get the connection string from the configuration file (app.config).

con = new SqlConnection(ConfigurationManager.ConnectionStrings["DBCon"].ConnectionString);

}

// Method to open the database connection if it's not already open or broken.

public SqlConnection OpenCon()

{

// Check if the connection state is Broken or Closed.

if (con.State == ConnectionState.Broken || con.State == ConnectionState.Closed)

{

// Open the connection if it's either broken or closed.

con.Open();

}

// Return the open connection.

return con;

}

// Method to close the database connection if it's open.

public void CloseCon()

{

// Check if the connection object is not null.

if (con != null)

{

// Check if the connection is currently open.

if (con.State == ConnectionState.Open)

{

// Close the connection if it's open.

con.Close();

}

}

}

}

}

* **data.cs**

namespace Hospital\_Managament\_System

{

internal class data : DAO

{

SqlDataAdapter da;

DataTable dt;

public void AddDoctor(string ne, string ad, string cy, int ag, string ph, string em, string se, string dt, decimal py)

{

// Create a SQL command object to execute a stored procedure.

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspNewDoctor"; // Specify the stored procedure name.

cmd.CommandType = System.Data.CommandType.StoredProcedure; // Indicate it's a stored procedure.

// Add parameters to the SQL command with corresponding values.

cmd.Parameters.AddWithValue("@ne", ne);

cmd.Parameters.AddWithValue("@ad", ad);

cmd.Parameters.AddWithValue("@cy", cy);

cmd.Parameters.AddWithValue("@ag", ag);

cmd.Parameters.AddWithValue("@ph", ph);

cmd.Parameters.AddWithValue("@em", em);

cmd.Parameters.AddWithValue("@se", se);

cmd.Parameters.AddWithValue("@dt", dt);

cmd.Parameters.AddWithValue("@py", py);

// Execute the command to insert the doctor into the database.

cmd.ExecuteNonQuery();

// Close the database connection after the operation.

CloseCon();

}

public void AddNurse(string ne, string ad, string cy, int ag, string ph, string em, string se, string dt, string hr, string ge)

{

SqlCommand cmd= OpenCon().CreateCommand();

cmd.CommandText = "uspNewNurse";

cmd.CommandType = System.Data.CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@ne", ne);

cmd.Parameters.AddWithValue("@ad", ad);

cmd.Parameters.AddWithValue("@cy", cy);

cmd.Parameters.AddWithValue("@ag", ag);

cmd.Parameters.AddWithValue("@ph", ph);

cmd.Parameters.AddWithValue("@em", em);

cmd.Parameters.AddWithValue("@se", se);

cmd.Parameters.AddWithValue("@dt", dt);

cmd.Parameters.AddWithValue("@hr", hr);

cmd.Parameters.AddWithValue("@ge", ge);

cmd.ExecuteNonQuery();

CloseCon();

}

public void AddPatient(string ne, string ad, string cy, int ag, string ph, string em, string pn)

{

// Create a SQL command object to execute a stored procedure.

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspNewPatient"; // Specify the stored procedure name.

cmd.CommandType = System.Data.CommandType.StoredProcedure; // Indicate it's a stored procedure.

// Add parameters to the SQL command with corresponding values.

cmd.Parameters.AddWithValue("@ne", ne);

cmd.Parameters.AddWithValue("@ad", ad);

cmd.Parameters.AddWithValue("@cy", cy);

cmd.Parameters.AddWithValue("@ag", ag);

cmd.Parameters.AddWithValue("@ph", ph);

cmd.Parameters.AddWithValue("@em", em);

cmd.Parameters.AddWithValue("@pn", pn);

// Execute the command to insert the patient into the database.

cmd.ExecuteNonQuery();

// Close the database connection after the operation.

CloseCon();

}

public void AddReceptionist(string ne, string ad, string cy, int ag, string ph, string em, string se, string dt, string st)

{

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspNewReceptionist"; // Stored procedure to add a receptionist.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

// Adding parameters to the stored procedure.

cmd.Parameters.AddWithValue("@ne", ne);

cmd.Parameters.AddWithValue("@ad", ad);

cmd.Parameters.AddWithValue("@cy", cy);

cmd.Parameters.AddWithValue("@ag", ag);

cmd.Parameters.AddWithValue("@ph", ph);

cmd.Parameters.AddWithValue("@em", em);

cmd.Parameters.AddWithValue("@se", se);

cmd.Parameters.AddWithValue("@dt", dt);

cmd.Parameters.AddWithValue("@st", st);

cmd.ExecuteNonQuery(); // Executes the stored procedure.

CloseCon(); // Closes the database connection.

}

public void AddTechnician(string ne, string ad, string cy, int ag, string ph, string em, string se, string dt, string ct)

{

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspNewTechnician"; // Stored procedure to add a technician.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

// Adding parameters to the stored procedure.

cmd.Parameters.AddWithValue("@ne", ne);

cmd.Parameters.AddWithValue("@ad", ad);

cmd.Parameters.AddWithValue("@cy", cy);

cmd.Parameters.AddWithValue("@ag", ag);

cmd.Parameters.AddWithValue("@ph", ph);

cmd.Parameters.AddWithValue("@em", em);

cmd.Parameters.AddWithValue("@se", se);

cmd.Parameters.AddWithValue("@dt", dt);

cmd.Parameters.AddWithValue("@ct", ct);

cmd.ExecuteNonQuery(); // Executes the stored procedure.

CloseCon(); // Closes the database connection.

}

public DataTable User()

{

da = new SqlDataAdapter(); // Initializes a data adapter.

dt = new DataTable(); // Initializes a data table.

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspUser"; // Stored procedure to fetch user data.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd; // Links the command to the data adapter.

da.Fill(dt); // Fills the data table with the results.

return dt; // Returns the data table containing user information.

}

public DataTable AllDoctors()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspAllDoctors"; // Stored procedure to fetch all doctors.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable AllNurses()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspAllNurses"; // Stored procedure to fetch all Nurses.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable AllPatients()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspAllPatients"; // Stored procedure to fetch all Patient.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable AllReceptionist()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspAllReceptionist"; // Stored procedure to fetch all Receptionist.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable AllTechnician()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspAllTechnician"; // Stored procedure to fetch all Technician.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable MaleNurses()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspMaleNurses"; // Stored procedure to fetch all Male Nurses.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable PatientsAbove25()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspPatientsAbove25"; // Stored procedure to fetch all Patients Above 25.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable TechnicianByDepartment()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspTechnicianByDepartment"; // Stored procedure to fetch all Technician By Department.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

public DataTable GetReceptionist()

{

da = new SqlDataAdapter();

dt = new DataTable();

SqlCommand cmd = OpenCon().CreateCommand();

cmd.CommandText = "uspGetReceptionist"; // Stored procedure to fetch all Get Receptionist.

cmd.CommandType = System.Data.CommandType.StoredProcedure;

da.SelectCommand = cmd;

da.Fill(dt); // Fills the data table with the query result.

return dt; // Returns all doctor records.

}

}

}

* **View.cs**

public partial class View : Form

{

DAO dao = new DAO(); // DAO instance to manage database connections.

data d = new data(); // Instance of the data class for executing database operations.

SqlDataAdapter da; // Used for data retrieval into DataTable.

DataTable dt; // Holds data retrieved from the database for binding to controls.

public View()

{

InitializeComponent(); // Initializes the form's components.

}

private void View\_Load(object sender, EventArgs e)

{

dgvHospital.DataSource = d.User(); // Loads the user data into the DataGridView when the form loads.

}

private void BtnSubmit\_Click(object sender, EventArgs e)

{

string selectquery = cboData.SelectedItem.ToString(); // Retrieves the selected query from the ComboBox.

dt = new DataTable();

switch (selectquery) // Executes a different method based on the selected option.

{

case "All Doctors":

dt = d.AllDoctors(); // Fetches all doctors.

break;

case "All Nurses":

dt = d.AllNurses(); // Fetches all nurses.

break;

case "All Patients":

dt = d.AllPatients(); // Fetches all patients.

break;

case "All Technician":

dt = d.AllTechnician(); // Fetches all technicians.

break;

case "Male Nurses":

dt = d.MaleNurses(); // Fetches all male nurses.

break;

case "Patients Above 25":

dt = d.PatientsAbove25(); // Fetches patients aged above 25.

break;

case "Technician By Department":

dt = d.TechnicianByDepartment(); // Fetches technicians by department.

break;

}

dgvHospital.DataSource = dt; // Updates the DataGridView with the selected query's data.

}

private void BtnRefresh\_Click(object sender, EventArgs e)

{

dgvHospital.DataSource = d.User(); // Reloads user data into the DataGridView.

}

private void BtnAR\_Click(object sender, EventArgs e)

{

dgvHospital.DataSource = d.AllReceptionist(); // Fetches and displays all receptionists.

}

private void BtnGR\_Click(object sender, EventArgs e)

{

dgvHospital.DataSource = d.GetReceptionist(); // Fetches a specific receptionist or filtered receptionist data.

}

private void BtnExit\_Click(object sender, EventArgs e)

{

Form1 form1 = new Form1(); // Instantiates the main form.

form1.ShowDialog(); // Displays the main form as a dialog box.

}

}

}

#### A screenshot of a computer AI-generated content may be incorrect.**Screenshot’s**

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

* A screenshot of a medical form

  AI-generated content may be incorrect.**A screenshot of a medical form

  AI-generated content may be incorrect.Queries**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a medical form

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a medical form

AI-generated content may be incorrect.A screenshot of a medical form

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

##### **Reference**

* Balaraman, P. and Kosalram, K. (2013). Information Engineering and Electronic Business. Information Engineering and Electronic Business, [online] (1), pp.50–58. Available at:

<https://www.mecs-press.org/ijieeb/ijieeb-v5-n1/IJIEEB-V5-N1-6.pdf>

* Google Books. (2025). The C# Programming Language (Covering C# 4.0).

[online] Available at:

<https://books.google.ie/books?hl=en&lr=&id=s-IH_x6ytuQC&oi=fnd&pg=PT13&dq=c%23+programming&ots=lv878PcReY&sig=UknqCl3-zypa_Jb7AtsqgMVd360&redir_esc=y#v=onepage&q=c%23%20programming&f=false>

* Google Books. (2022). Understanding the New SQL.

[online] Available at:

<https://books.google.ie/books?hl=en&lr=&id=ZOOMSTZ4T_QC&oi=fnd&pg=PA1&dq=sql&ots=e3KceEQ8bB&sig=MpHZh66GbzAjerFEofywLfrZUyo&redir_esc=y#v=onepage&q=sql&f=false>

* Koç, H., Erdoğan, A.M., Barjakly, Y. and Peker, S. (2021). UML Diagrams in Software Engineering Research: A Systematic Literature Review.

Proceedings, [online] 74(1), p.13.

doi: <https://doi.org/10.3390/proceedings2021074013>.

‌

‌