Hospital Management System

Course: IE322 – Computer Applications in Industrial Engineering

Instructor: Dr. Muhammad Atif Shahzad

Semester: Spring 2025

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Members | ID | Picture of the Member |
| M1 | Saad Alharbi | 2340647 | A person with a mustache and beard  AI-generated content may be incorrect. |
| M2 | Hussien Alharbi | 2340604 | A person wearing a headdress and a white shirt  AI-generated content may be incorrect. |
| M3 | Ibrahim Alsulami | 2340593 | A person wearing a red and white headdress  AI-generated content may be incorrect. |
| M4 | Nawaf Alghamdi | 2339001 |  |
| M5 | Faris Ghazzawi | 2339847 | A person wearing a white robe  AI-generated content may be incorrect. |

Table of Contents

[Part I: Project Overview 4](#_Toc197249774)

[Introduction 4](#_Toc197249775)

[Objectives 4](#_Toc197249776)

[Workflow Diagram 5](#_Toc197249777)

[Setting up the C# Application 6](#_Toc197249778)

[Part II: System Design & Code 7](#_Toc197249779)

[Setting up SQL Database 7](#_Toc197249780)

[Classes Overview 8](#_Toc197249781)

[Form Layouts & Functions 10](#_Toc197249782)

[11](#_Toc197249783)

[11](#_Toc197249784)

[Part III: Application Output 14](#_Toc197249785)

[Part IV: Testing & Evaluation 14](#_Toc197249786)

[Testing 14](#_Toc197249787)

[Conclusion & Future Work 14](#_Toc197249788)

[Part V: Team & References 15](#_Toc197249789)

[Work Distribution 15](#_Toc197249790)

# Part I: Project Overview

## Introduction

The Hospital Management System is a desktop program created to simplify the handling of patient data in a medical setting. The system's goal is to replace conventional paper-based techniques with a digital solution that is more effective, safe, and easily accessible. Its core features, which include user authentication, patient registration, diagnosis input, and historical data retrieval, were built with C# Windows Forms and integrated with SQL Server. The system provides a multi-panel interface that integrates database systems and software engineering concepts, allowing users to manage medical records, enter diagnoses, and examine full patient histories in an easy-to-use workflow. Its architecture shows how object-oriented design and relational database integration can be used practically in a real-world setting.

## Objectives

The following objectives are driving the development of this system:

* Get practical experience using C# to create a Windows Forms application.
* Recognize and put into practice session control and user authentication using a login form.
* Create and implement graphical user interfaces (GUI) to gather and handle patient data.
* Create and connect SQL tables for patients and diagnoses using the concepts of database design.
* To combine and display patient and diagnosis data in a single historical view, use SQL JOIN operations.

## Workflow Diagram

The Hospital Management System's workflow offers organized and easy-to-use user experience, stepping the user through each step of the application's functionality. The process centers around a dashboard that can be accessed through login and offers access to the system's main functions.

Description of the System Flow   
  
**Login Form:**

* To gain access to the system, the user must enter correct login information.
* The user's password and username are the same, grants access.

**Navigating the dashboard:**

* Add Patient: Adds new patient information to the Add Patient table.
* Diagnosis: Enters and saves diagnosis data for current patients in the Patient More table.
* View History: Integrates information from both tables and shows it in a DataGrid View display.
* Exit: Ends the program.

**Backend Interaction:**

* A diagram of a diagram

  AI-generated content may be incorrect.SQL Server database communication and input validation are included in every panel operation.

Figure 1- Workflow Diagram

## Setting up the C# Application

* A new **Windows Forms App (.NET Framework)** project was created.

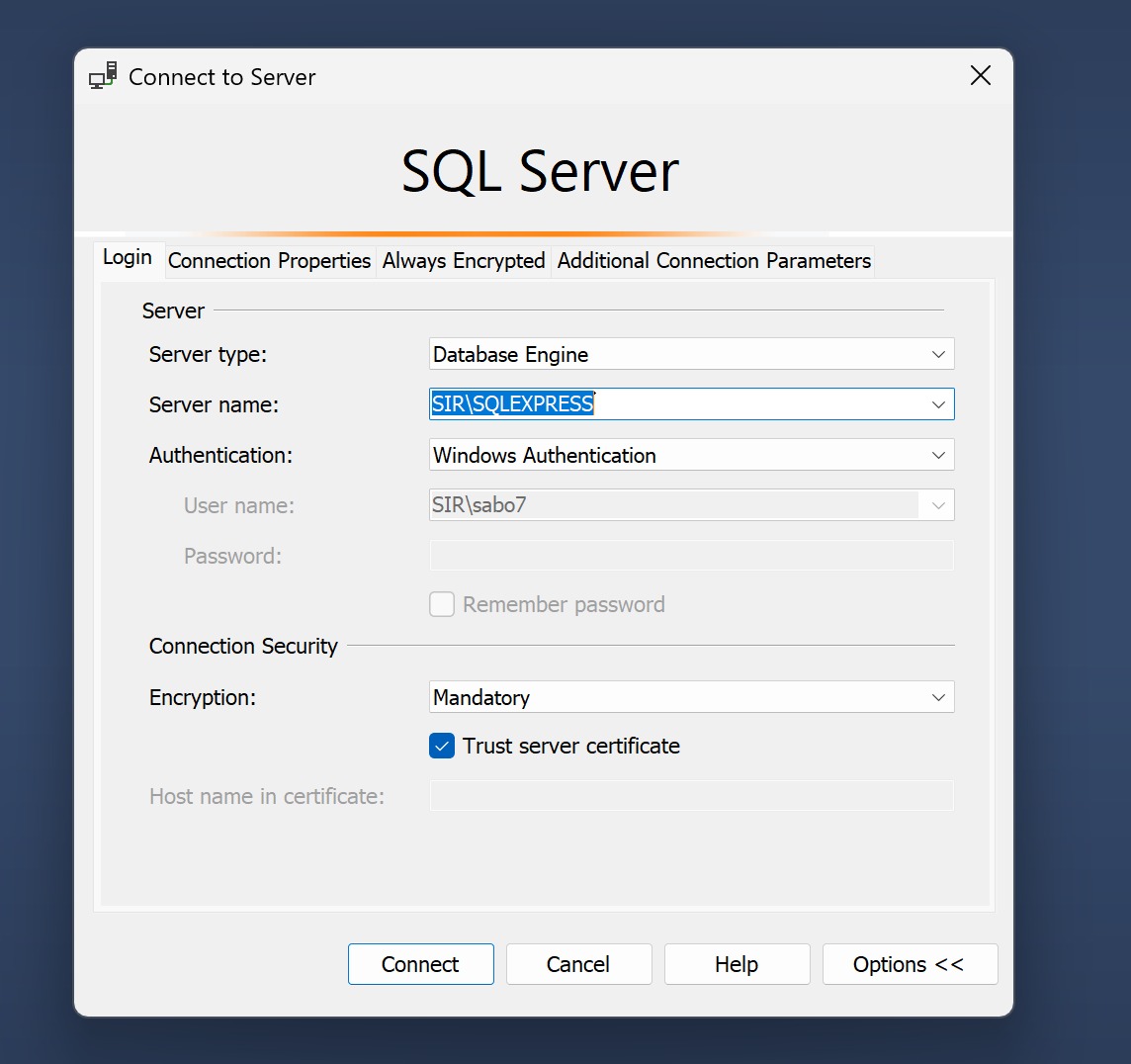
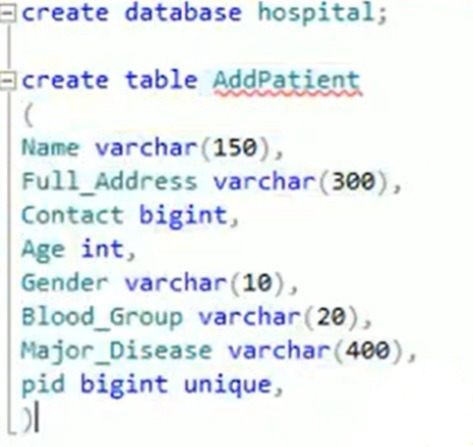


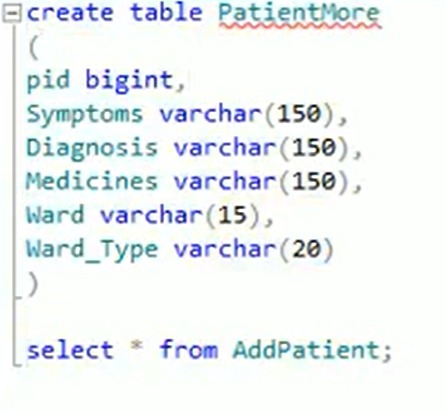
We installed **Microsoft SQL Server Management Studio (SSMS)** to create and manage the database locally.

* **Database Name:** Hospital
* Tables were created and populated directly in SSMS.

# Part II: System Design & Code

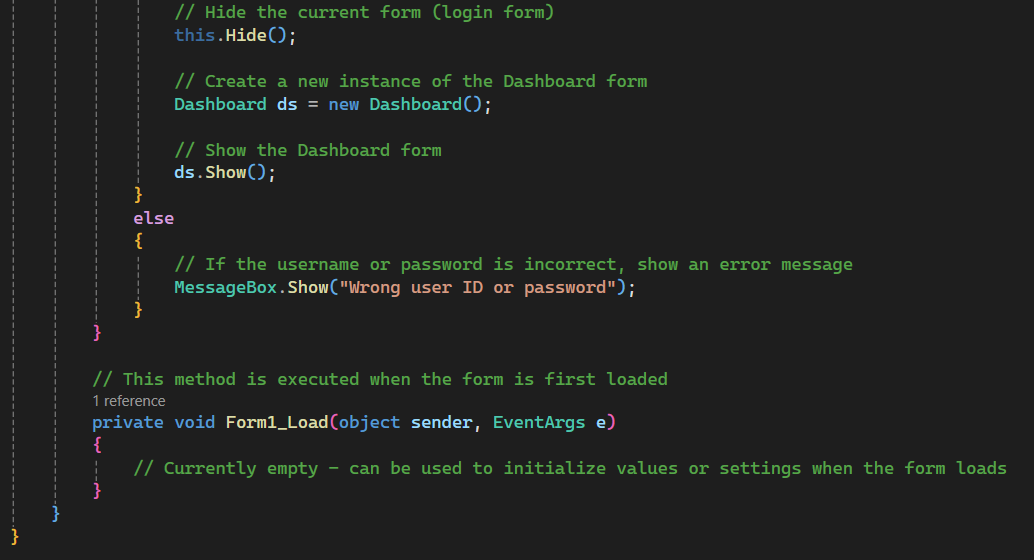
## Setting up SQL Database





The Hospital database was created using SQL Server by first executing the CREATE DATABASE statement to establish the Hospital database. Two tables were then created: AddPatient to store general patient information (including fields such as Name, Age, Gender, and Contact), where pid was set as the primary key to uniquely identify each patient, and PatientMore to store diagnosis-related data (such as Symptoms, Diagnosis, Medicines, and Ward), with pid serving as a foreign key linking each diagnosis record to a corresponding patient in AddPatient. The pid column in AddPatient ensures uniqueness and consistency for patient records, while the foreign key in PatientMore maintains referential integrity between the two tables. Data types such as BIGINT for pid and Contact, VARCHAR for text fields like Name and Diagnosis, and INT for numeric fields like Age were used to properly store the data. The relationship between the two tables is a one-to-one connection, enforced through the use of the foreign key constraint on the pid column in PatientMore, ensuring that each diagnosis is linked to a valid patient.

## Classes Overview

A white rectangular object with black text

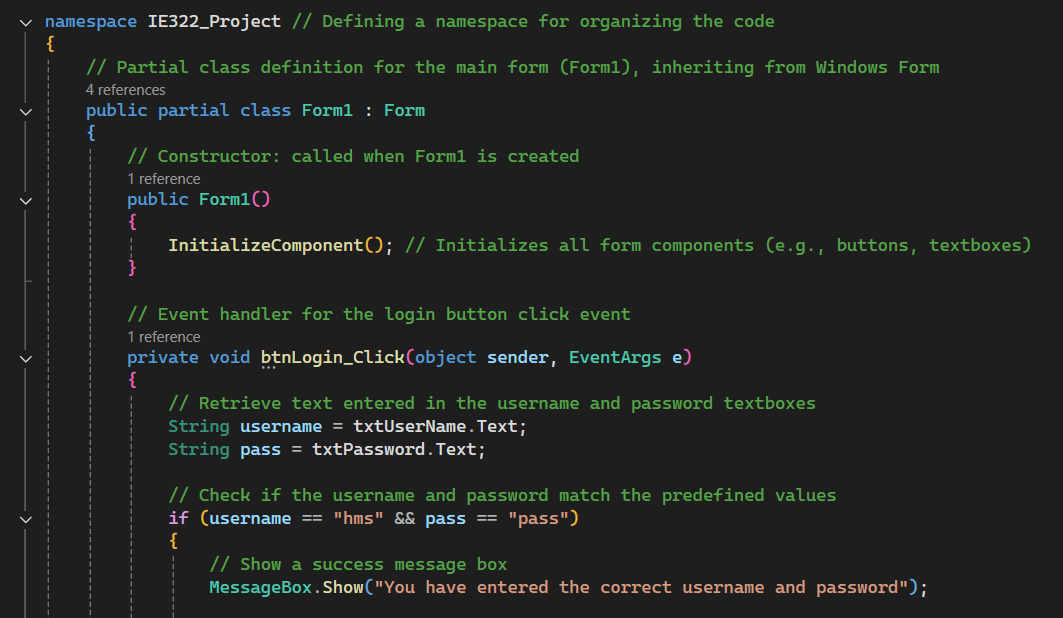
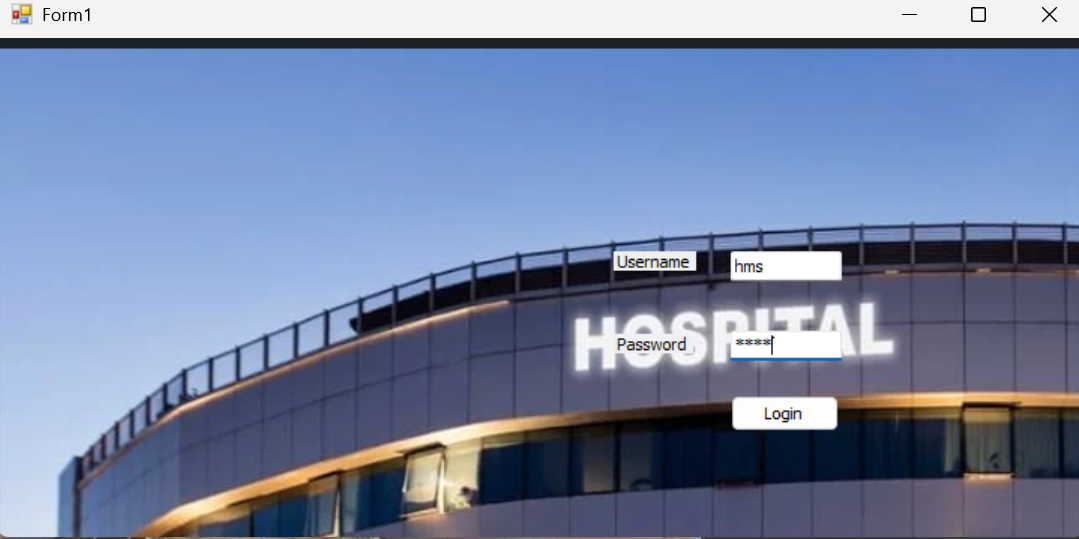
AI-generated content may be incorrect.

Figure 1

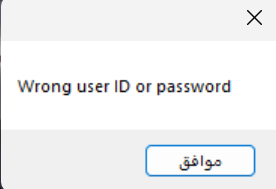


 Figure 2

In Form1, the login functionality is handled by the btnLogin\_Click event, which retrieves the username and password entered by the user in the respective textboxes. It then checks if the entered credentials match predefined values ("hms" and "pass"). If the credentials are correct, a success message shown in Figure 1 is displayed, the login form is hidden using this.Hide(), and a new instance of the Dashboard form is created and displayed using ds.Show(). If the credentials are incorrect, an error message shown in Figure 2 is displayed to the user indicating that the username or password is wrong.

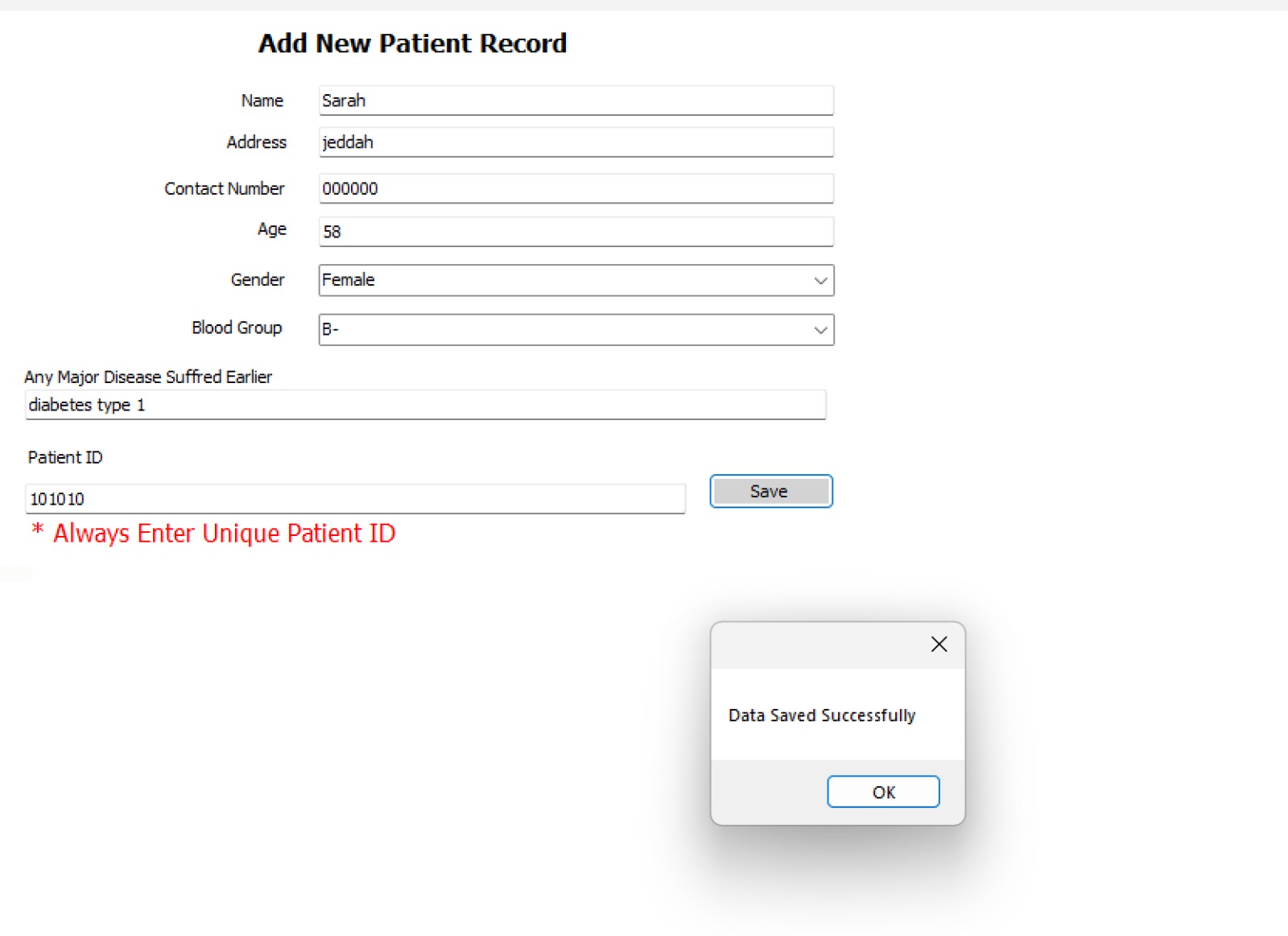
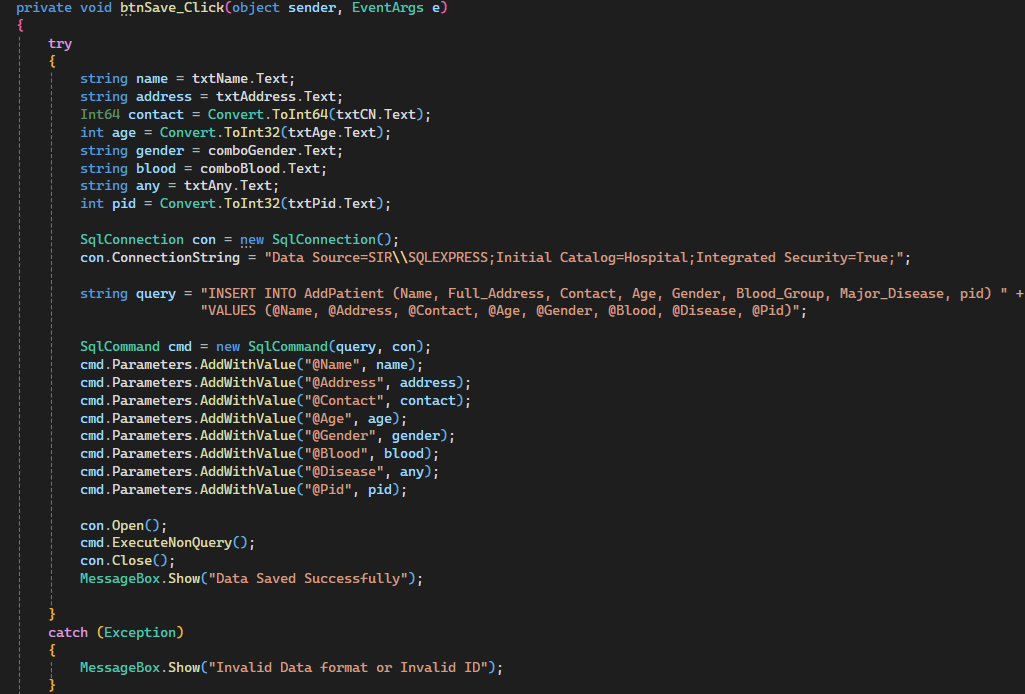


A screen shot of a computer program

AI-generated content may be incorrect.

In the Dashboard form, panel navigation is controlled through button click events like btnAddPatient\_Click, btnAddDiagnosis\_Click, and btnFullHistory\_Click. These events change the visibility of the panels by setting the Visible property of panel1, panel2, and panel4 accordingly. When the "Add Patient" button is clicked, panel2 is displayed, and the others are hidden; similarly, clicking "Add Diagnosis" shows panel1, and "Full History" shows panel4. The form’s constructor calls InitializeComponent(), which sets up all the UI components (such as buttons, textboxes, and panels) and their properties, ensuring the form is properly initialized when it is first loaded. Each event also handles additional actions, such as querying the database in the case of viewing patient history, and updating indicator label colors to provide feedback to the user about which section is active.

## Form Layouts & Functions

**Add Patient section**

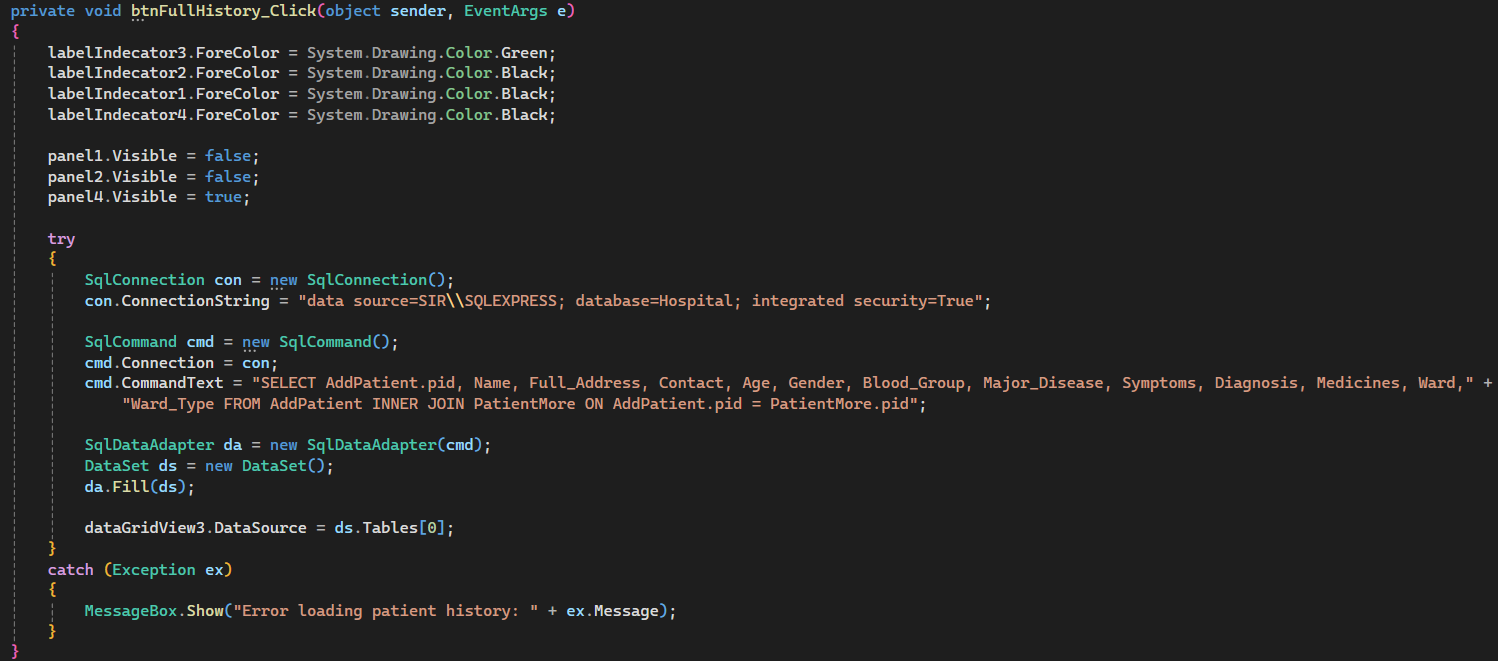
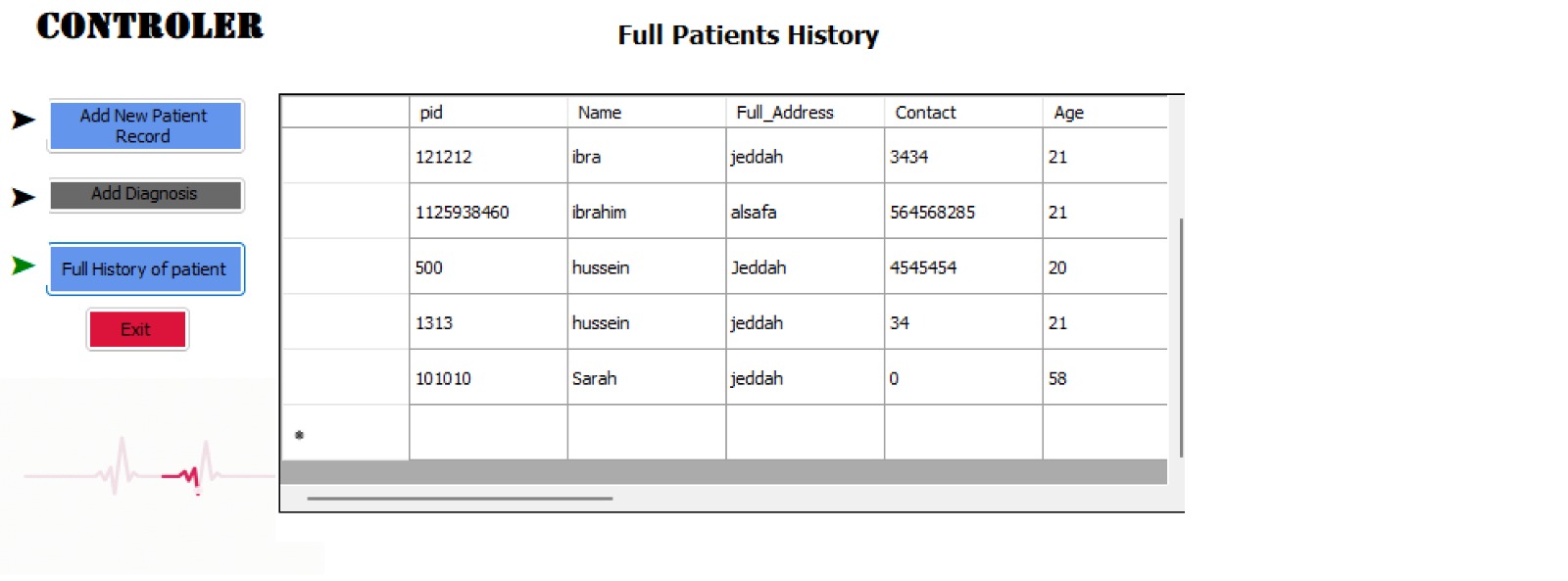
In the Add Patient section of the Dashboard, the user inputs patient details using TextBox controls for fields such as Name, Address, Contact, Age, Major Disease, and Patient ID, and ComboBox controls for selecting Gender and Blood Group. When the user clicks the Save button (btnSave), the btnSave\_Click event is triggered, which retrieves the entered data from the textboxes and combo boxes, and inserts it into the AddPatient table in the database using a parameterized SQL query to prevent SQL injection. The query is executed through a SqlConnection and SqlCommand, where the parameters are added with cmd.Parameters.AddWithValue(). If the data is saved successfully, a success message is shown, otherwise, an error message is displayed if the data format is invalid or if there’s an issue with the database connection.

**Diagnosis section**

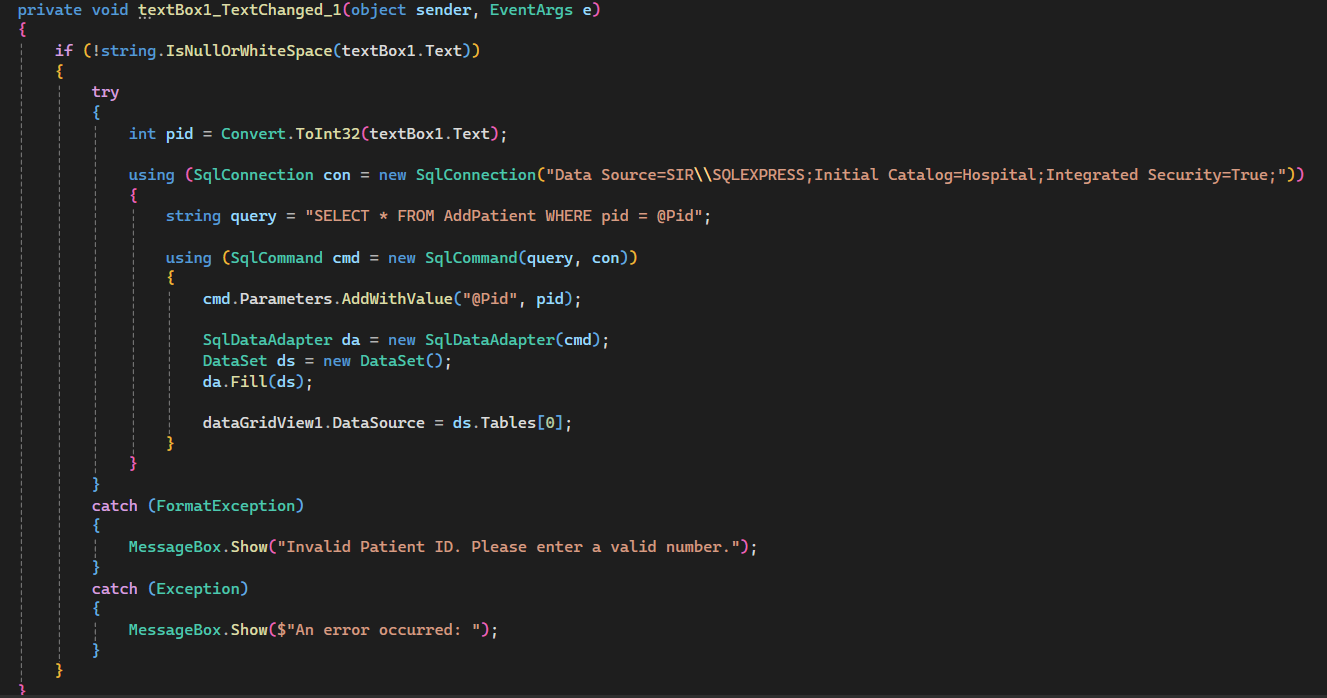
## 

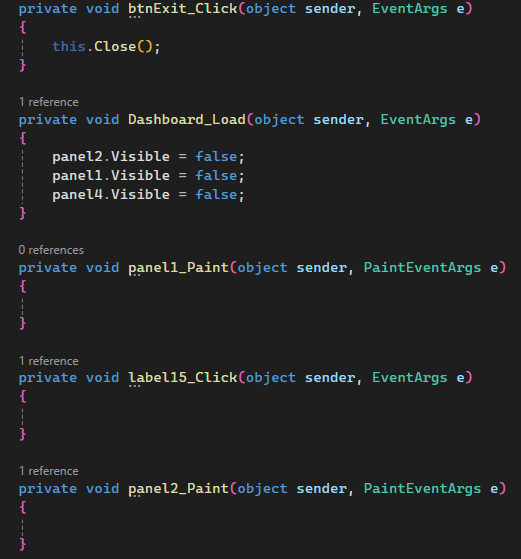
## A screenshot of a computer

In the Diagnosis section of the Dashboard, the user inputs details such as Patient ID (PID), Symptoms, Diagnosis, Medicines, and Ward using TextBox and ComboBox controls. When the user clicks the Save button (button1), the button1\_Click event is triggered, which collects the entered data from the textboxes and combo boxes and inserts it into the PatientMore table in the database using a parameterized SQL query. The query uses SqlConnection to connect to the database and SqlCommand to execute the query with the parameters added via cmd.Parameters.AddWithValue(). After successfully saving the diagnosis data, the form fields are cleared, and a success message is displayed. If there’s an error (e.g., invalid PID or database issues), an error message is shown to the user.

**History Viewer section**

In the History Viewer section of the Dashboard, the application uses a DataGridView control to display patient records by fetching joined data from the AddPatient and PatientMore tables using an SQL INNER JOIN query. This query combines patient details (such as Name, Contact, Age, Blood Group) from AddPatient with diagnosis-related information (like Symptoms, Diagnosis, Medicines) from PatientMore, using the pid (Patient ID) to match corresponding rows. The fetched data is then displayed in the DataGridView (dataGridView3). The query is executed within a try-catch block to handle potential errors, and if the operation is successful, the data is shown in the viewer; otherwise, an error message is displayed to the user.

This code is part of the View Patient History, triggered when the user types a Patient ID (PID) into textBox1. It converts the entered PID into an integer and then constructs an SQL query to fetch patient data from the AddPatient table based on the entered PID. The query is executed using a parameterized SQL command to prevent SQL injection, and the resulting data is displayed in a DataGridView (dataGridView1). If the user enters an invalid ID or non-numeric value, an error message is shown to inform them to input a valid Patient ID. If any other errors occur, a general error message is displayed.



The btnExit\_Click method closes the form when the Exit button is clicked. The Dashboard\_Load method hides panel1, panel2, and panel4 when the form is first loaded, likely to display them later based on user actions. The panel1\_Paint and panel2\_Paint methods are placeholders triggered when panel1 and panel2 are painted, but they currently do not contain any logic for custom drawing or UI changes. Similarly, the label15\_Click method is triggered when label15 is clicked, but it is empty and does not perform any action. These methods are likely set up for further functionality, depending on the needs of the application.

# Part III: Application Output

The hospital management system developed in C# is designed to simplify the process of managing patient records and medical history. Once a user logs into the system, they are presented with a dashboard that allows them to perform a variety of functions. These include creating a new patient file, filling out the patient's personal information such as name, ID number, age, gender, contact details, and medical history. After entering the data, the user can save the record, add diagnoses for the current visit, and review any past medical information related to the patient.  
  
The main output of this application is its graphical user interface (GUI), which save the data of the patient. These outputs ensure that the user knows when an operation has been successfully completed. The design of the interface focuses on clarity and ease of use, ensuring that healthcare staff can quickly navigate the application.

## Part IV: Testing & Evaluation

## Testing

To ensure the reliability and accuracy of the hospital management system, we conducted thorough manual testing across all critical functions. Each key feature of the system was tested using both valid and invalid inputs. For instance, the login functionality was tested using both correct and incorrect credentials. When the correct username and password were entered, the application correctly redirected the user to the main dashboard. On the other hand, when incorrect credentials were used, the application displayed an appropriate error message.  
  
We also tested the patient creation form by entering complete data and saving it. The system confirmed the entry with a success message. When we attempted to save a patient form with missing required fields (like name or age), the system refused to proceed and prompted the user to complete the necessary information. Additionally, we tested the diagnosis feature by adding multiple diagnoses and ensuring they were properly stored and retrieved later in the patient’s medical history.  
  
All tested features behaved as expected, and no major bugs or crashes occurred during the process. The application handled edge cases well, such as handling blank inputs, duplicate patient entries, and invalid data formats.

## Conclusion & Future Work

In conclusion, the hospital management application successfully provides core functionalities required for managing patient data in a clinical setting. Through comprehensive manual testing, the system demonstrated stable performance, accurate data handling, and user-friendly interface design. Its ability to register patients, track medical histories, and manage diagnoses forms a solid foundation for further development.  
  
For future enhancements, implementing user roles (admin, doctor, nurse) can strengthen access control and workflow efficiency. Additional features like appointment scheduling, search functionality, and medical report generation could significantly boost usability. A mobile version using NET MAUI could also increase flexibility and accessibility for healthcare professionals on the move.

# Part V: Team & References

## Work Distribution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task | M1 | M2 | M3 | M4 | M5 |
| Form Code |  |  |  | \* | \* |
| Dashboard Code | \* | \* | \* |  |  |
| UI Design | \* | \* | \* | \* | \* |
| SGL Database | \* |  |  |  |  |
| Report | \* | \* | \* | \* | \* |
| GitHub | \* |  |  |  |  |

Githup Link:  
https://github.com/Sir0SaaD/saadie322.git