# STUDENT ENROLLMENT SYSTEM

**PROJECT BY-**Aniket Nimbalkar

## Table of Contents:

## 1. Introduction

The Student Enrollment System is a software application developed to simplify and manage the process of enrolling students in courses within an educational institution. The system enables administrators to maintain student records, handle course enrollments, track payments, and log all actions performed in the system. It also provides a simple and secure interface for administrators to manage the entire student lifecycle.

## Objective

- The primary objectives of the Student Enrollment System are:
- To manage student data and enrollment information.
- To allow administrators to add, edit, delete, and search student records.
- To enable course assignments and enrollment tracking.
- To handle payment processing and record tracking.
- To maintain an audit trail for every action performed by administrators.

## 2. System Overview

The Student Enrollment System consists of several key components:

1. **Admin Dashboard:** Provides administrators with a comprehensive view of student data, course enrollment, and payment records.

2. **Student Enrollment Form:** Form used to input or update student details, select courses, and record enrollment dates.

3. **Database:** Stores student records, course data, payment information, and audit logs.

4. **Payment Handling:** Interface to track payments related to student enrollments.

# 3. System Design

## 3.1 Database Design

The system relies on a relational database (SQL Server) for storing and managing data. Here is the schema for the key tables:

**Students Table**

Stores student personal information and enrollment data.

| Column Name | Data Type | Description |
|---|---|---|
| StudentId | INT | Primary key, auto-increment |
| FirstName | VARCHAR(100) | Student's first name |
| LastName | VARCHAR(100) | Student's last name |
| DOB | DATETIME | Date of birth |
| Gender | VARCHAR(10) | Gender of the student |
| EnrollmentDate | DATETIME | Date the student enrolled |
| CourseId | INT | Foreign key to the Courses table |

**Payments Table**

This table logs payments made by students.

| Column Name | Data Type | Description |
|---|---|---|
| PaymentId | INT | Primary key, auto-increment |
| StudentId | INT | Foreign key to the Students table |
| Amount | DECIMAL(10, 2) | Amount paid |
| PaymentDate | DATETIME | Date of payment |
| PaymentMethod | VARCHAR(50) | PaymentMethod |

**Courses Table**

Holds details about courses.

| Column Name | Data Type | Description |
|---|---|---|
| CourseId | INT | Primary key, auto-increment |
| CourseName | VARCHAR(100) | Name of the course |
| Description | TEXT | Course description |

**AuditLogs Table**

Tracks administrator actions (add, edit, delete) on student data.

| Column Name | Data Type | Description |
|---|---|---|
| ActionId | INT | Primary key, auto-increment |
| ActionType | VARCHAR(50) | Type of action (e.g., 'Add', 'Update') |
| TableName | VARCHAR(100) | Table affected (e.g., 'Students') |
| RecordId | INT | ID of the record affected |
| UserName | VARCHAR(100) | Name of the user performing the action |
| Timestamp | DATETIME | Timestamp when the action was performed |

**StudentCoursesTable**

Handles the many-to-many relationship between students and courses.

| Column Name | Data Type | Description |
|---|---|---|
| StudentCourseId | INT | Primary key, auto-increment |
| StudentId | INT | Foreign key to Students table |
| CourseId | INT | Foreign key to Courses table |

### 3.2 System Flow

The flow of the system involves several steps for both administrators and the system itself:

1. Login: The admin logs in using credentials.
2. Admin Dashboard: Post-login, the admin can:
    - View all students.
    - Search and filter student records.
    - Add new students or edit existing records.
3. Student Enrollment Form:
    - Admin inputs student details (first name, last name, gender, DOB).
    - Selects available courses for the student.
4. Payment Handling: Admin can log payments related to student enrollments.
5. Audit Log: Every action performed by the admin (e.g., adding, editing, or deleting students) is logged for auditing purposes.

### 3.3 Technology Stack

The following technologies were used to build the system:

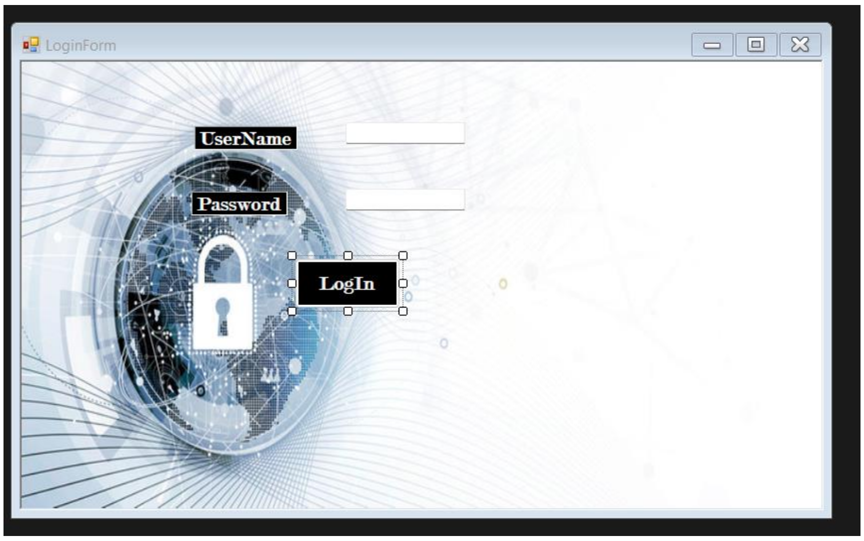
- **Frontend:**
    - C# Windows Forms for user interfaces.
    - Visual Studio for development and design.
- **Backend:**
    - SQL Server for database management.
    - SQL Server Management Studio (SSMS) for managing the database.
- **Tools:**
    - Visual Studio for development.
    - SQL Server Management Studio (SSMS) for the database.

# 4. Implementation

## 4.1 Login Form

The Login Form is the authentication interface for the administrator. The form collects the admin's username and password, validates them against stored credentials, and grants access to the system if successful.

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string username = txtUsername.Text.Trim();
    string password = txtPassword.Text.Trim();
    if (username == "admin" && password == "123")
    {
        MessageBox.Show("Login Successful!");
        this.Hide();  // Hide the login form
        AdminDashboardForm dashboard = new AdminDashboardForm();
        dashboard.Show();  // Show the Admin Dashboard form
    }
    else
    {
        MessageBox.Show("Invalid username or password. Please try again.");
    }
}
```



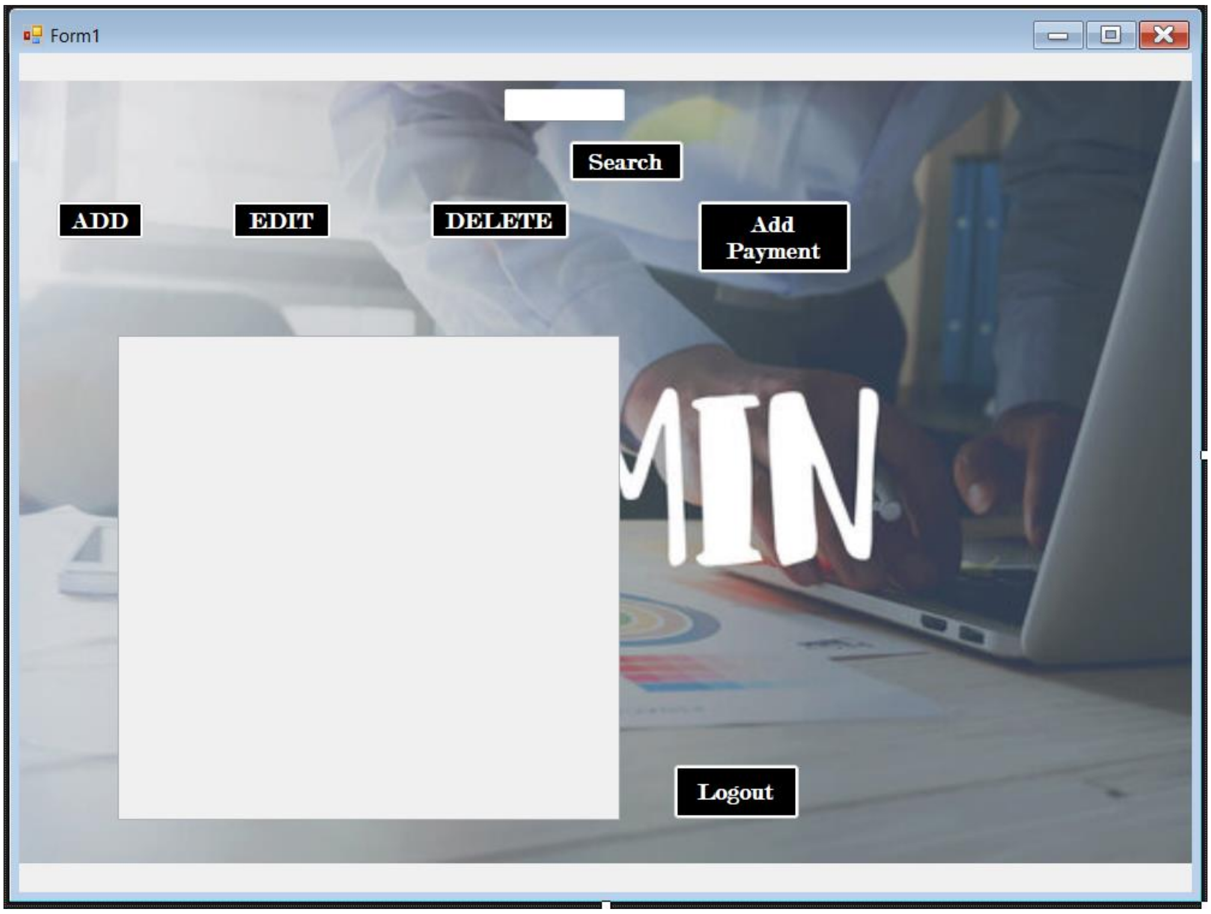
### 4.2 Admin Dashboard Form

The Admin Dashboard allows administrators to manage the student records. It provides a data grid for viewing, adding, editing, and deleting student records.

**Sample Code for Loading Student Data:**

**C#**

```
 private void LoadStudentData(int studentId)
 {
    string query = "SELECT * FROM Students WHERE StudentId = @StudentId";

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
       SqlCommand cmd = new SqlCommand(query, conn);
       cmd.Parameters.AddWithValue("@StudentId", studentId);
       conn.Open();

       SqlDataReader reader = cmd.ExecuteReader();
       if (reader.Read())
       {
          txtFirstName.Text = reader["FirstName"].ToString();
          txtLastName.Text = reader["LastName"].ToString();
          dateTimePickerDOB.Value = Convert.ToDateTime(reader["DOB"]);
          cmbGender.SelectedItem = reader["Gender"].ToString();
          cmbCourse.SelectedValue = reader["CourseId"];
          dateTimePickerEnrollmentDate.Value = Convert.ToDateTime(reader["EnrollmentDate"]);
    }
}
```



**4.3 Student Enrollment Form**

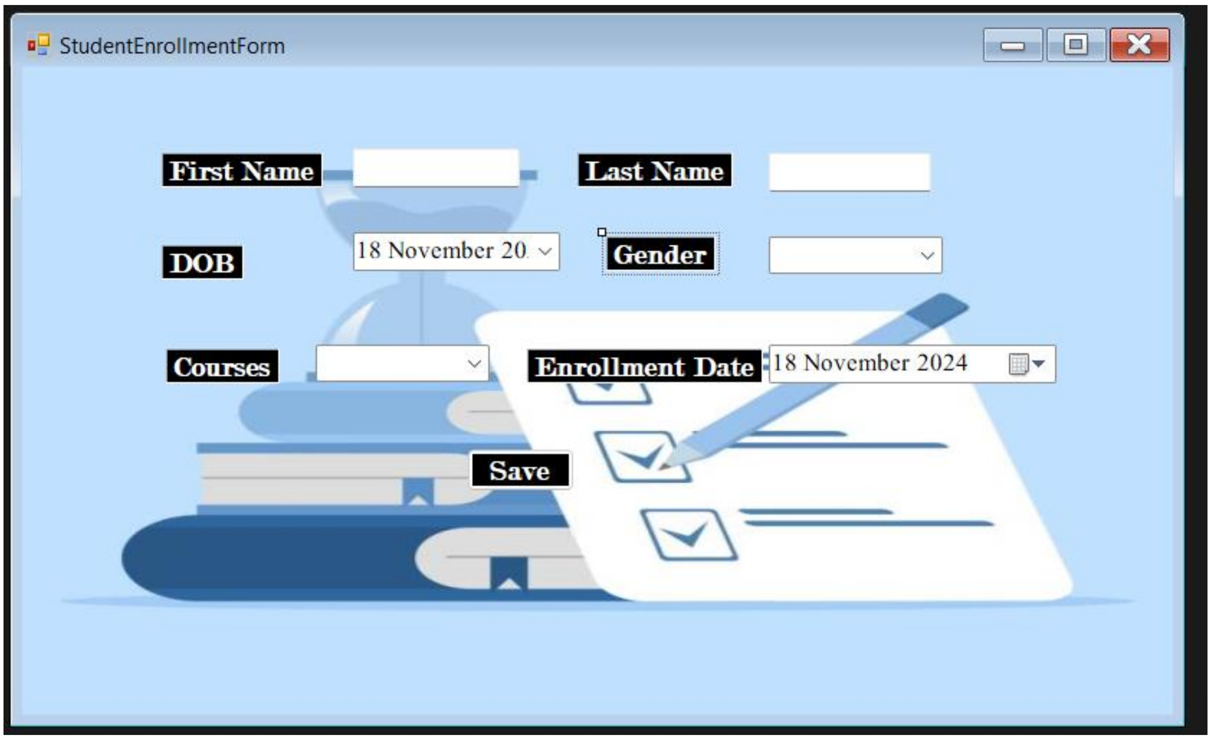
The Student Enrollment Form allows administrators to input or update student information and enroll students in courses.

**Sample Code for Saving Student Information:**

**C#**

```
private void btnSave_Click(object sender, EventArgs e)

 {

    string firstName = txtFirstName.Text.Trim();

    string lastName = txtLastName.Text.Trim();

    DateTime dob = dateTimePickerDOB.Value;

    string gender = cmbGender.SelectedItem?.ToString();

    DateTime enrollmentDate = dateTimePickerEnrollmentDate.Value;
    cmbCourse.SelectedValue.ToString();
    int courseId = Convert.ToInt32(cmbCourse.SelectedValue);
    if (cmbCourse.SelectedValue == null)
    {
        MessageBox.Show("Please select a course.");
        return;
    }

    if (string.IsNullOrEmpty(firstName) || string.IsNullOrEmpty(lastName) ||
string.IsNullOrEmpty(gender) || courseId == 0)
    {
        MessageBox.Show("Please fill in all the fields.");
        return;
    }
    string query = "";
    if (studentId > 0)
    {
        query = "UPDATE Students SET FirstName = @FirstName, LastName = @LastName, DOB =
@DOB, Gender = @Gender, EnrollmentDate=@EnrollmentDate, " +
                "CourseId = @CourseId WHERE StudentId = @StudentId";
    }
    else
    {
        query = "INSERT INTO Students (FirstName, LastName, DOB, Gender, CourseId,EnrollmentDate)
VALUES (@FirstName, @LastName, @DOB, @Gender, @CourseId,@EnrollmentDate)";
    }
    MessageBox.Show($"Saving student: {firstName} {lastName}, Gender: {gender}, DOB:
{dob.ToShortDateString()}, Course ID: {courseId},
EnrollmentDate:{enrollmentDate.ToShortDateString()}");

    try
    {
```

```
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@FirstName", firstName);
        cmd.Parameters.AddWithValue("@LastName", lastName);
        cmd.Parameters.AddWithValue("@DOB", dob);
        cmd.Parameters.AddWithValue("@Gender", gender);
        cmd.Parameters.AddWithValue("@CourseId", courseId);
        cmd.Parameters.AddWithValue("@EnrollmentDate", enrollmentDate);

        if (studentId > 0)
        {
            cmd.Parameters.AddWithValue("@StudentId", studentId);
        }
        conn.Open();
        int rowsAffected = cmd.ExecuteNonQuery();
        if (rowsAffected > 0)
        {
            MessageBox.Show("Student saved successfully!");
            this.Close();
        }
        else
        {
            MessageBox.Show("An error occurred while saving the student.");
        }
        DateTime enrollmentdate = dateTimePickerEnrollmentDate.Value;
    }
}
catch (Exception ex)
{
    MessageBox.Show("Error: " + ex.Message);
    }
}
```



**4.4 Database Connection and Query Execution**

The SQL Server database is connected using SqlConnection, and queries are executed using SqlCommand.

**Sample Code for Database Connection:**

**C#**

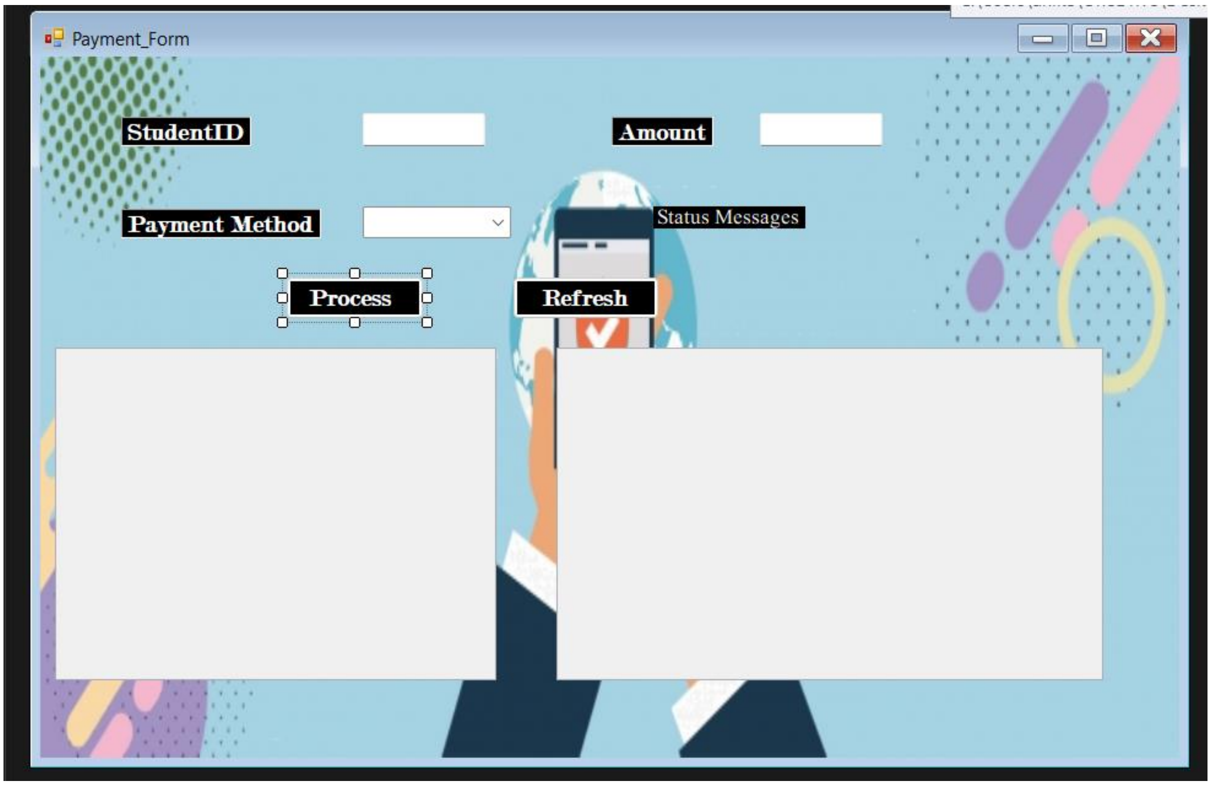
```
using (SqlConnection conn = new SqlConnection(connectionString))
{
    SqlCommand cmd = new SqlCommand(query, conn);
    conn.Open();
    cmd.ExecuteNonQuery();
}
```

**4.3 Payment Form**

The **Payment Form** is used by the admin to record payments made by students for their enrolled courses.

```
private void ProcessPayment(int studentId, decimal amount, string paymentMethod)
```

```
{
    // Insert Payment record
    string paymentQuery = "INSERT INTO Payments (StudentId, Amount, PaymentDate, 
PaymentMethod, Status) " +
                          "VALUES (@StudentId, @Amount, @PaymentDate, @PaymentMethod, @Status)";

    // Log action in AuditLogs table
    LogAuditAction("Insert", "Payments", studentId, "Admin", $"Processed payment of {amount} for 
StudentId: {studentId}");

    try
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();

            using (SqlCommand cmd = new SqlCommand(paymentQuery, conn))
            {

                cmd.Parameters.AddWithValue("@StudentId", studentId);
                cmd.Parameters.AddWithValue("@Amount", amount);
                cmd.Parameters.AddWithValue("@PaymentDate", DateTime.Now);
                cmd.Parameters.AddWithValue("@PaymentMethod", paymentMethod);
                cmd.Parameters.AddWithValue("@Status", "Successful");

                // Execute the query
                cmd.ExecuteNonQuery();

                lblMessage.Text = "Payment processed successfully!";
                lblMessage.ForeColor = System.Drawing.Color.Green;
            }
        }
    }
    catch (Exception ex)
    {
        lblMessage.Text = "Error processing payment: " + ex.Message;
        lblMessage.ForeColor = System.Drawing.Color.Red;
    }
}
```



## 4.6 Logging and Audit

Every action performed by the admin is logged in the AuditLogs table. The LogAuditAction method captures the type of action, the table affected, and the timestamp.

**Sample Code for Logging Actions:**

**C#**

```
private void LogAuditAction(string actionType, string tableName, int recordId, string userName,
string actionMessage)

{

    string query = "INSERT INTO AuditLogs (ActionType, TableName, RecordId, UserName,
ActionMessage, Timestamp) " +
 "VALUES (@ActionType, @TableName, @RecordId, @UserName, @ActionMessage, @Timestamp)";
  using (SqlConnection conn = new SqlConnection(connectionString))
  {
    SqlCommand cmd = new SqlCommand(query, conn);
    cmd.Parameters.AddWithValue("@ActionType", actionType);
    cmd.Parameters.AddWithValue("@TableName", tableName);
    cmd.Parameters.AddWithValue("@RecordId", recordId);
    cmd.Parameters.AddWithValue("@UserName", userName);
    cmd.Parameters.AddWithValue("@ActionMessage", actionMessage);
    cmd.Parameters.AddWithValue("@Timestamp", DateTime.Now);
    conn.Open();
    cmd.ExecuteNonQuery();
  }
}
```

## 5. Testing and Evaluation

**Testing Methods:**

- Unit Testing: Each module was tested individually (e.g., database operations, form inputs).
- Integration Testing: Validated the interaction between different components (frontend and backend).
- User Acceptance Testing (UAT): Feedback was gathered from test users to evaluate the user interface and functionality.

**Test Results:**

- All functional tests passed successfully.
- No security vulnerabilities were identified.
- The user interface was easy to use and intuitive.

## 6. Challenges and Solutions

### 6.1 Database Errors

  - **Issue:** "Invalid column name" error during execution.
  - **Solution:** Ensure that the database schema matches the application code by running migration scripts**.**

### 6.2 Null Values

- **Issue:** Null values in mandatory fields.
  - **Solution**: Implement validation checks to ensure that required fields are not left blank.

## 7. Conclusion

The Student Enrollment System provides an efficient solution for managing student enrollments, tracking course , handling payments, and maintaining audit logs. The system is user-friendly, secure, and fully functional for administrative tasks.

## 8. Future Enhancements

- User Authentication: Implement role-based access control for different user types (admin, students).
- Student Portal: Develop a portal for students to view their enrolled courses, grades, and payments.
- Advanced Reporting: Add features to generate detailed reports on student performance, course enrollments, and payments.