Table of Contents

[Introduction 1](#_Toc168542256)

[Program Files 1](#_Toc168542257)

[Database Schema 1](#_Toc168542258)

[SqlConnection.cs 5](#_Toc168542259)

[Program.cs 5](#_Toc168542260)

[Admin.cs 8](#_Toc168542261)

[Teacher.cs 19](#_Toc168542262)

[Student.cs 30](#_Toc168542263)

[Functionalities 33](#_Toc168542264)

[Admin Functionalities 33](#_Toc168542265)

[Teacher Functionalities 34](#_Toc168542266)

[Student Functionalities 34](#_Toc168542267)

# Introduction

The School Management System is a console based application that manages data for administrators, teachers, and students. It allow its users to log in based on their roles and perform various tasks such as view grades, their attendance, and subjects assigned to them.

# Program Files

## Database Schema

A well defined database schema is the backbone of any application. MySql database was used to store and access the data required for this application.

DROP DATABASE IF EXISTS SchoolDB;

CREATE DATABASE IF NOT EXISTS SchoolDB;

USE SchoolDB;

-- Create Teachers table

CREATE TABLE IF NOT EXISTS Teachers (

TeacherID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

ContactNumber VARCHAR(20),

Address VARCHAR(100),

Email VARCHAR(100) UNIQUE,

Password VARCHAR(50)

);

-- Create Administrators table

CREATE TABLE IF NOT EXISTS Administrators (

AdminID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

ContactNumber VARCHAR(20),

Address VARCHAR(100),

Email VARCHAR(100) UNIQUE,

Password VARCHAR(50)

);

-- Create Students table

CREATE TABLE IF NOT EXISTS Students (

StudentID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

ContactNumber VARCHAR(20),

Address VARCHAR(100),

Email VARCHAR(100) UNIQUE,

Password VARCHAR(50),

EnrollmentDate DATE

);

-- Create Classes table

CREATE TABLE IF NOT EXISTS Classes (

ClassID INT PRIMARY KEY,

ClassName VARCHAR(50),

GradeLevel INT

);

-- Create Subjects table

CREATE TABLE IF NOT EXISTS Subjects (

SubjectID INT PRIMARY KEY,

SubjectName VARCHAR(100),

Credits INT

);

-- Create TeacherSubjects junction table

CREATE TABLE IF NOT EXISTS TeacherSubjects (

TeacherSubjectID INT AUTO\_INCREMENT PRIMARY KEY,

TeacherID INT,

SubjectID INT,

FOREIGN KEY (TeacherID) REFERENCES Teachers(TeacherID),

FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)

);

-- Create StudentSubjects junction table

CREATE TABLE IF NOT EXISTS StudentSubjects (

StudentSubjectID INT AUTO\_INCREMENT PRIMARY KEY,

StudentID INT,

SubjectID INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)

);

-- Create Attendance table

CREATE TABLE IF NOT EXISTS Attendance (

AttendanceID INT AUTO\_INCREMENT PRIMARY KEY,

StudentID INT,

SubjectID INT,

Date DATE,

Status VARCHAR(10),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)

);

-- Create Grades table

CREATE TABLE IF NOT EXISTS Grades (

GradeID INT AUTO\_INCREMENT PRIMARY KEY,

StudentID INT,

SubjectID INT,

GradeValue DECIMAL(5,2),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)

);

-- Insert values into Teachers table

INSERT INTO Teachers (TeacherID, FirstName, LastName, ContactNumber, Address, Email, Password)

VALUES

(1, 'Ram', 'Thapa', '9800000001', 'Kirtipur, Kathmandu', 'ram.thapa@gmail.com', 'thapa123'),

(2, 'Sita', 'Gurung', '9800000002', 'Lalitpur', 'sita.gurung@gmail.com', 'gurung123'),

(3, 'Hari', 'Shrestha', '9800000003', 'Bhaktapur', 'hari.shrestha@gmail.com', 'shrestha123');

-- Insert values into Administrators table

INSERT INTO Administrators (AdminID, FirstName, LastName, ContactNumber, Address, Email, Password)

VALUES

(1, 'Gita', 'Pandey', '9800000004', 'Pokhara', 'gita.pandey@gmail.com', 'pandey123');

-- Insert values into Students table

INSERT INTO Students (StudentID, FirstName, LastName, ContactNumber, Address, Email, Password, EnrollmentDate)

VALUES

(1, 'Amit', 'Karki', '9800000005', 'Butwal', 'amit.karki@gmail.com', 'karki123', '2024-01-01'),

(2, 'Bina', 'Shah', '9800000006', 'Dharan', 'bina.shah@gmail.com', 'shah123', '2024-01-02'),

(3, 'Chandra', 'Rai', '9800000007', 'Hetauda', 'chandra.rai@gmail.com', 'rai123', '2024-01-03');

-- Insert values into Classes table

INSERT INTO Classes (ClassID, ClassName, GradeLevel)

VALUES

(1, 'Class 1', 1),

(2, 'Class 2', 2),

(3, 'Class 3', 3);

-- Insert values into Subjects table

INSERT INTO Subjects (SubjectID, SubjectName, Credits)

VALUES

(1, 'Mathematics', 3),

(2, 'Science', 4),

(3, 'English', 2),

(4, 'History', 3);

-- Insert values into Grades table

INSERT INTO Grades (StudentID, SubjectID, GradeValue)

VALUES

(1, 1, 85.50), (1, 2, 78.00), (1, 3, 92.30),

(2, 2, 88.75), (2, 3, 69.50), (2, 4, 74.25),

(3, 3, 81.00), (3, 4, 77.25);

-- Insert values into Attendance table

INSERT INTO Attendance (StudentID, SubjectID, Date, Status)

VALUES

(1, 1, '2024-05-01', 'Present'), (1, 2, '2024-05-02', 'Absent'), (1, 3, '2024-05-03', 'Present'),

(2, 2, '2024-05-01', 'Present'), (2, 3, '2024-05-02', 'Present'), (2, 4, '2024-05-03', 'Absent'),

(3, 3, '2024-05-01', 'Absent'), (3, 4, '2024-05-02', 'Present');

-- Insert values into TeacherSubjects table

INSERT INTO TeacherSubjects (TeacherID, SubjectID)

VALUES

(1, 1), (1, 2), (1, 3),

(2, 2), (2, 3), (2, 4),

(3, 3), (3, 4);

-- Insert values into StudentSubjects table

INSERT INTO StudentSubjects (StudentID, SubjectID)

VALUES

(1, 1), (1, 2), (1, 3),

(2, 2), (2, 3), (2, 4),

(3, 3), (3, 4);

## SqlConnection.cs

This class is responsible for creating connection with the database. As we frequently need to access tables so instead of repeating the same piece of code again and again, assigning it a class and accessing it through its method would help to avoid redundancy of same code.

## using MySql.Data.MySqlClient;

## namespace ManagementSystem

## {

## public class SqlConnection

## {

## private static string connectionString = "datasource=127.0.0.1;port=3306;username=root;password=;database=DETAILS;";

## public static MySqlConnection GetConnection()

## {

## return new MySqlConnection(connectionString);

## }

## }

## }

## Program.cs

This class is responsible for authenticating user based on their role i.e administrator or student or teacher.

## using Dapper;

## namespace ManagementSystem

## {

## class Program

## {

## public static void Main(String[] args)

## {

## Console.WriteLine("╔════════════════════════════════════════════════════╗");

## Console.WriteLine("║ School Management System - Login Page ║");

## Console.WriteLine("╚════════════════════════════════════════════════════╝");

## Console.WriteLine(" 1. Admin");

## Console.WriteLine(" 2. Teacher");

## Console.WriteLine(" 3. Student");

## Console.Write("Enter your role: ");

## int role = Convert.ToInt32(Console.ReadLine());

## Console.Write("Enter your username: ");

## String username = Console.ReadLine();

## Console.Write("Enter your password: ");

## String password = Console.ReadLine();

## //String connectionString = "datasource=127.0.0.1;port=3306;username=root;password=;database=DETAILS;";

## try

## {

## using (var connect = SqlConnection.GetConnection())

## {

## connect.Open();

## if (role == 1)

## {

## // returns the number of rows from admin table where username and password matched

## string query = "SELECT COUNT(\*) FROM ADMIN WHERE username = @Username and password = @Password";

## // assign parameter to @username and @password in query

## var parameters = new { Username = username, Password = password };

## // if the match is found count will be 1 else it will be 0

## int count = connect.QuerySingleOrDefault<int>(query, parameters);

## if (count > 0)

## {

## Admin admin = new Admin();

## }

## else

## {

## Console.WriteLine("Invalid Username or Password");

## }

## }

## else if (role == 2)

## {

## // returns the number of rows from teacher table where username and password matches

## string query = "SELECT COUNT(\*) FROM TEACHER WHERE username = @Username and password = @Password";

## var parameters = new { Username = username, Password = password };

## int count = connect.QuerySingleOrDefault<int>(query, parameters);

## if (count > 0)

## {

## Teacher teacher = new Teacher(username);

## 

## }

## else

## {

## Console.WriteLine("Invalid Username or Password");

## }

## }

## else if (role == 3)

## {

## // returns the number of rows from student table where username and password matches

## string query = "SELECT COUNT(\*) FROM STUDENT WHERE username = @Username and password = @Password";

## var parameters = new { Username = username, Password = password };

## int count = connect.QuerySingleOrDefault<int>(query, parameters);

## if (count > 0)

## {

## Student student = new Student(username);

## }

## else

## {

## Console.WriteLine("Invalid Username or Password");

## }

## }

## else

## {

## Console.WriteLine("Invalid LogIn Credentials Provided!!");

## }

## }

## }

## catch(Exception ex)

## {

## Console.WriteLine("Unexpected Error:");

## Console.WriteLine(ex.Message);

## }

## Console.WriteLine("Press Enter key to exit...");

## Console.ReadLine();

## }

## }

## }

## Admin.cs

using Dapper;

namespace ManagementSystem

{

class Program

{

public static void Main(String[] args)

{

Console.WriteLine("╔════════════════════════════════════════════════════╗");

Console.WriteLine("║ School Management System - Login Page ║");

Console.WriteLine("╚════════════════════════════════════════════════════╝");

Console.WriteLine(" 1. Admin");

Console.WriteLine(" 2. Teacher");

Console.WriteLine(" 3. Student");

Console.Write("Enter your role: ");

int role = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter your username: ");

String username = Console.ReadLine();

Console.Write("Enter your password: ");

String password = Console.ReadLine();

//String connectionString = "datasource=127.0.0.1;port=3306;username=root;password=;database=DETAILS;";

try

{

using (var connect = SqlConnection.GetConnection())

{

connect.Open();

if (role == 1)

{

// returns the number of rows from admin table where username and password matched

string query = "SELECT COUNT(\*) FROM ADMIN WHERE username = @Username and password = @Password";

// assign parameter to @username and @password in query

var parameters = new { Username = username, Password = password };

// if the match is found count will be 1 else it will be 0

int count = connect.QuerySingleOrDefault<int>(query, parameters);

if (count > 0)

{

Admin admin = new Admin();

}

else

{

Console.WriteLine("Invalid Username or Password");

}

}

else if (role == 2)

{

// returns the number of rows from teacher table where username and password matches

string query = "SELECT COUNT(\*) FROM TEACHER WHERE username = @Username and password = @Password";

var parameters = new { Username = username, Password = password };

int count = connect.QuerySingleOrDefault<int>(query, parameters);

if (count > 0)

{

Teacher teacher = new Teacher(username);

}

else

{

Console.WriteLine("Invalid Username or Password");

}

}

else if (role == 3)

{

// returns the number of rows from student table where username and password matches

string query = "SELECT COUNT(\*) FROM STUDENT WHERE username = @Username and password = @Password";

var parameters = new { Username = username, Password = password };

int count = connect.QuerySingleOrDefault<int>(query, parameters);

if (count > 0)

{

Student student = new Student(username);

}

else

{

Console.WriteLine("Invalid Username or Password");

}

}

else

{

Console.WriteLine("Invalid LogIn Credentials Provided!!");

}

}

}

catch(Exception ex)

{

Console.WriteLine("Unexpected Error:");

Console.WriteLine(ex.Message);

}

Console.WriteLine("Press Enter key to exit...");

Console.ReadLine();

}

}

}

## Teacher.cs

using Dapper;

namespace ManagementSystem

{

public class Teacher

{

string loggedInUsername;

public Teacher(string username)

{

while (true)

{

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine(" Management System ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine("1. Student Management ");

Console.WriteLine(" 11. View Students Assigned ");

Console.WriteLine(" 12. Search Students ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine("2. Class Management ");

Console.WriteLine(" 21. View Class Assigned ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine("3. Subject Management ");

Console.WriteLine(" 31. View Subjects Assigned ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine("4. Marks Management ");

Console.WriteLine(" 41. Assign Marks ");

Console.WriteLine(" 42. Update Marks ");

Console.WriteLine(" 43. View Marks ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine("5. Attendance Management ");

Console.WriteLine(" 51. Attendance for a Class ");

Console.WriteLine(" 52. Update Attendance ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine(" 0. Exit ");

Console.WriteLine("══════════════════════════════════════════");

loggedInUsername = username;

Console.Write("Enter a Option: ");

int userResponse = Convert.ToInt32(Console.ReadLine());

if (userResponse == 11)

{

int teacherid = IdConverter(loggedInUsername);

List<string> subjects = SubjectAssigned(teacherid);

Console.WriteLine();

// loop through the list of subjects that are assigned to teacher

foreach (string sub in subjects)

{

Console.WriteLine($"\t \t {sub}: ");

Console.WriteLine("{0, -15} | {1, -15} | {2, -15} | {3, -30} | {4, -30}", "Student ID", "First Name", "Last Name", "Contact Number", "Address");

Console.WriteLine(new string('-', 109));

// stores student details for the subject passed as argument

List<Student> students = DisplayStudents(sub);

foreach (var student in students)

{

Console.WriteLine("{0, -15} | {1, -15} | {2, -15} | {3, -30} | {4, -30}", student.StudentID, student.FirstName, student.LastName, student.ContactNumber, student.Address);

Console.WriteLine();

}

}

}

else if (userResponse == 12)

{

var connect = SqlConnection.GetConnection();

Console.WriteLine("Enter student ID to search: ");

int studentId = Convert.ToInt32(Console.ReadLine());

string studentIdExistsQuery = "SELECT StudentID from Student Where studentID = @studentID";

var parameterForStudentIdExistsQuery = new { studentID = studentId };

var student = connect.Query<int>(studentIdExistsQuery, parameterForStudentIdExistsQuery);

if (student.Contains(studentId))

{

SearchStudent(studentId);

}

else

{

Console.WriteLine("Student details not found.");

}

}

else if (userResponse == 31)

{

// returns teacher id

int teacherid = IdConverter(loggedInUsername);

// returns the list of subjects assigned

List<string> subjects = SubjectAssigned(teacherid);

foreach (var sub in subjects)

{

Console.WriteLine("\t" + sub);

}

}

else if (userResponse == 41)

{

while (true)

{

var connect = SqlConnection.GetConnection();

Console.Write("Enter Student ID: ");

int studentId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Subject ID: ");

int subjectId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Mark: ");

decimal markValue = Convert.ToDecimal(Console.ReadLine());

string MarksassignQuery = "INSERT INTO Marks (StudentID, SubjectID, MarkValue) VALUES (@StudentID, @SubjectID, @MarkValue)";

var parameterForMarksassignQuery = new { StudentID = studentId, SubjectID = subjectId, MarkValue = markValue };

connect.Execute(MarksassignQuery, parameterForMarksassignQuery);

Console.WriteLine("");

string continueToAdd = Console.ReadLine();

if (continueToAdd != "YES" || continueToAdd != "yes")

{

break;

}

}

}

else if (userResponse == 42)

{

var connect = SqlConnection.GetConnection();

Console.Write("Enter Student ID: ");

int studentId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Subject ID: ");

int subjectId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Mark: ");

decimal markValue = Convert.ToDecimal(Console.ReadLine());

string markUpdateQuery = "UPDATE Marks Set MarkValue = @MarkValue WHERE StudentID = @StudentID AND SubjectID = @SubjectID";

var parameterForMarkUpdateQuery = new { MarkValue = markValue, StudentID = studentId, SubjectID = subjectId };

connect.Execute(markUpdateQuery, parameterForMarkUpdateQuery);

}

else if (userResponse == 43)

{

var connect = SqlConnection.GetConnection();

Console.Write("Enter Student ID: ");

int studentId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Subject ID: ");

int subjectId = Convert.ToInt32(Console.ReadLine());

string markVieWQuery = "SELECT MarkValue FROM Marks WHERE StudentID= @StudentID AND SubjectID= @SubjectID";

var parameterForMarkViewQuery = new { StudentID = studentId, SubjectID = subjectId };

var mark = connect.QueryFirstOrDefault<float>(markVieWQuery, parameterForMarkViewQuery);

if (mark != 0)

{

Console.WriteLine($"Marks for the subject code: {subjectId} and Student ID: {studentId} is {mark}");

}

else

{

Console.WriteLine("Invalid StudentId or SubjectID");

}

}

else if (userResponse == 51)

{

var connect = SqlConnection.GetConnection();

int teacherId = IdConverter(loggedInUsername);

List<string> subjectsAssigned = SubjectAssigned(teacherId);

for (int i = 1; i <= subjectsAssigned.Count(); i++)

{

Console.WriteLine($"{i}. {subjectsAssigned[i - 1]}");

}

Console.Write("Select the subject for attendance: ");

int subjectSelection = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("{0, -15} | {1, -15} | {2, -15} | {3, -30} | {4, -30}", "Student ID", "First Name", "Last Name", "Contact Number", "Address");

var students = DisplayStudents(subjectsAssigned[subjectSelection - 1]);

string subjectIdQuery = "SELECT SubjectID FROM Subject WHERE SubjectName = @SubjectName";

int subjectId = connect.QuerySingleOrDefault<int>(subjectIdQuery, new { SubjectName = subjectsAssigned[subjectSelection - 1] });

foreach (var student in students)

{

Console.WriteLine("{0, -15} | {1, -15} | {2, -15} | {3, -30} | {4, -30}", student.StudentID, student.FirstName, student.LastName, student.ContactNumber, student.Address);

Console.Write("Attendence Status (Absent/Present): ");

string attendenceStatus = Console.ReadLine().ToUpper();

string attendanceAddQuery = "INSERT INTO Attendance (StudentID, SubjectID, Date, Status) VALUES (@StudentID, @SubjectID, @Date, @Status)";

var parameterForAttendanceAddQuery = new { StudentID = student.StudentID, SubjectID = subjectId, Date = DateTime.Now, Status = attendenceStatus };

connect.Execute(attendanceAddQuery, parameterForAttendanceAddQuery);

}

}

else if (userResponse == 52)

{

var connect = SqlConnection.GetConnection();

Console.Write("Enter the student id: ");

int studentId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the subject id: ");

int subjectId = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the date of attendence (YYy-mm-dd:");

DateTime Date = Convert.ToDateTime(Console.ReadLine());

Console.Write("Enter the status: ");

string status = Console.ReadLine();

string attendanceUpdateQuery = "UPDATE Attendance SET Status = @Status WHERE StudentID = @StudentID AND SubjectID = @SubjectID AND Date = @Date";

var parameterForAttendanceUpdateQuery = new { Status = status, StudentID = studentId, SubjectID = subjectId, Date = Date };

connect.Execute(attendanceUpdateQuery, parameterForAttendanceUpdateQuery);

}

else if (userResponse == 0)

{

break;

}

}

// returns teacherid of the currentlyloggedin user

static int IdConverter(string loggedInUsername)

{

var connect = SqlConnection.GetConnection();

// stores a sql query that returns teacherid based on the username provided (logged in username)

String query = "SELECT TeacherID FROM Teacher WHERE Username = @Username";

// assign username of currently logged in teacher to the username variable

var parameters = new { Username = loggedInUsername };

// returns teacherid of the currently logged in teacher

int teacherid = connect.QuerySingleOrDefault<int>(query, parameters);

return teacherid;

}

// returns a list of subject assigned to a teacher

static List<string> SubjectAssigned(int teacherid)

{

var connect = SqlConnection.GetConnection();

// joins subject and teacher subject table and return list of subjectname

string subjectAssignedQuery = "SELECT s.SubjectName, s.SubjectID " + "FROM Subject s " + "JOIN TeacherSubject ts ON s.SubjectID = ts.SubjectID " + "WHERE ts.TeacherID = @TeacherID";

var parametersForSubjectAssignedQuery = new { TeacherID = teacherid };

var subjects = connect.Query<string>(subjectAssignedQuery, parametersForSubjectAssignedQuery);

return new List<string>(subjects);

}

static void SearchStudent(int studentId)

{

var connect = SqlConnection.GetConnection();

string studentSearchQuery = "SELECT StudentID, FirstName, LastName, ContactNumber, Address FROM Student WHERE StudentID = @StudentID";

var parameterForStudentSearchQuery = new { StudentID = studentId };

var studentDetails = connect.Query<StudentDetails>(studentSearchQuery, parameterForStudentSearchQuery);

Console.WriteLine("{0, -15} | {1, -15} | {2, -15} | {3, -30} | {4, -30}", "Student ID", "First Name", "Last Name", "Contact Number", "Address");

Console.WriteLine(new string('-', 109));

Console.WriteLine();

foreach (var student in studentDetails)

{

Console.WriteLine("{0, -15} | {1, -15} | {2, -15} | {3, -30} | {4, -30}", student.StudentID, student.FirstName, student.LastName, student.ContactNumber, student.Address);

Console.WriteLine();

}

}

// return list of students enrolled in each subject

static List<Student> DisplayStudents(string subjectname)

{

var connect = SqlConnection.GetConnection();

// join three tables: student, studentsubject and subject

string displayStudentsQuery = "SELECT s.StudentID, s.Firstname, s.Lastname, s.ContactNumber, s.Address " + "From Student s " + "JOIN StudentSubject ss ON s.StudentID = ss.StudentID " + "JOIN Subject sb ON ss.SubjectID = sb.SubjectID " + "WHERE sb.SubjectName = @Subjectname";

var parametersForDisplayStudentsQuery = new { Subjectname = subjectname };

// returns multiple columns of student table and map it to student object

var students = connect.Query<Student>(displayStudentsQuery, parametersForDisplayStudentsQuery).ToList();

return students;

}

}

// dummy class to store the list of students returned by DisplayStudents Method

private class Student

{

public int StudentID { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string ContactNumber { get; set; }

public string Address { get; set; }

}

// dummy class to store the student details returned

private class StudentDetails

{

public int StudentID { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string ContactNumber { get; set; }

public string Address { get; set; }

}

}

}

## Student.cs

using Dapper;

namespace ManagementSystem

{

public class Student

{

private readonly string \_username;

// Constructor to initialize the username and display the menu

public Student(string username)

{

\_username = username;

// Display the menu in a loop until the user chooses to exit

while (true)

{

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine(" Student Menu ");

Console.WriteLine("══════════════════════════════════════════");

Console.WriteLine("1. View Attendance ");

Console.WriteLine("2. View Marks ");

Console.WriteLine("3. View Percentage ");

Console.WriteLine("4. View Subjects Assigned ");

Console.WriteLine(" 0. Exit ");

Console.WriteLine("══════════════════════════════════════════");

Console.Write("Enter an Option: ");

int userResponse = Convert.ToInt32(Console.ReadLine());

// Call the appropriate method based on the user's choice

switch (userResponse)

{

case 1:

ViewAttendance();

break;

case 2:

ViewMarks();

break;

case 3:

ViewPercentage();

break;

case 4:

ViewSubjectsAssigned();

break;

case 5:

ConditionForNull(userResponse);

break;

case 0:

return; // Exit the loop if the user chooses to exit

default:

Console.WriteLine("Invalid choice."); // Inform the user of an invalid choice

break;

}

}

}

// Method to view attendance records

private void ViewAttendance()

{

var connect = SqlConnection.GetConnection(); // Establish a database connection

string query = "SELECT a.Date, a.Status FROM Attendance a " +

"JOIN Student s ON a.StudentID = s.StudentID " +

"WHERE s.Username = @Username";

var parameters = new { Username = \_username };

var attendanceRecords = connect.Query(query, parameters);

Console.WriteLine("Attendance Records:");

foreach (var record in attendanceRecords)

{

Console.WriteLine($"Date: {record.Date}, Status: {record.Status}");

}

}

// Method to view marks records

private void ViewMarks()

{

var connect = SqlConnection.GetConnection(); // Establish a database connection

string query = "SELECT sub.SubjectName, m.MarkValue FROM Marks m " +

"JOIN Student s ON m.StudentID = s.StudentID " +

"JOIN Subject sub ON m.SubjectID = sub.SubjectID " +

"WHERE s.Username = @Username";

var parameters = new { Username = \_username };

var marksRecords = connect.Query(query, parameters);

Console.WriteLine("Marks Records:");

foreach (var record in marksRecords)

{

Console.WriteLine($"Subject: {record.SubjectName}, Marks: {record.MarkValue}");

}

}

// Method to view percentage records

private void ViewPercentage()

{

var connect = SqlConnection.GetConnection(); // Establish a database connection

string query = "SELECT SUM(m.MarkValue) AS TotalMarks, COUNT(m.SubjectID) AS TotalSubjects " +

"FROM Marks m " +

"JOIN Student s ON m.StudentID = s.StudentID " +

"WHERE s.Username = @Username";

var parameters = new { Username = \_username };

var result = connect.QuerySingle(query, parameters);

decimal totalMarks = result.TotalMarks;

decimal totalSubjects = result.TotalSubjects;

decimal percentage = (totalMarks / totalSubjects);

Console.WriteLine("Percentage:");

Console.WriteLine($"Total Marks: {totalMarks}, Total Subjects: {totalSubjects}, Percentage: {percentage:F2}%");

}

// Method to view subjects assigned to the student

private void ViewSubjectsAssigned()

{

var connect = SqlConnection.GetConnection(); // Establish a database connection

string query = "SELECT sub.SubjectName FROM StudentSubject ss " +

"JOIN Student s ON ss.StudentID = s.StudentID " +

"JOIN Subject sub ON ss.SubjectID = sub.SubjectID " +

"WHERE s.Username = @Username";

var parameters = new { Username = \_username };

var subjectsRecords = connect.Query(query, parameters);

Console.WriteLine("Subjects Assigned:");

foreach (var record in subjectsRecords)

{

Console.WriteLine($"Subject: {record.SubjectName}");

}

}

private void ConditionForNull(int userResponse)

{

if (userResponse == null)

{

Console.WriteLine("Please Enter a valid option.");

}

}

}

}

# Functionalities

## Admin Functionalities

* **Add Teacher**: Allows administrators to add a new teacher to the system.
* **Delete Teacher**: Allows administrators to delete an existing teacher from the system.
* **Add Student**: Allows administrators to add a new student to the system.
* **Delete Student**: Allows administrators to delete an existing student from the system.
* **Add Subject**: Allows administrators to add a new subject to the system.
* **Delete Subject**: Allows administrators to delete an existing subject from the system.
* **Assign Subjects to Teacher**: Allows administrators to assign subjects to a teacher.
* **Assign Class for Subjects**: Allows administrators to assign classes for subjects.
* **Generate Student Performance Report**: Allows administrators to generate a report of student performance.
* **Generate Attendance Report**: Allows administrators to generate a report of student attendance.
* **Generate Teacher Performance Report**: Allows administrators to generate a report of teacher performance based on average grades.

## Teacher Functionalities

* **View Students Assigned**: Allows teachers to view the list of students assigned to their subjects.
* **Search Students**: Allows teachers to search for a specific student by ID.
* **View Class Assigned**: Allows teachers to view the list of classes assigned to them.
* **View Subjects Assigned**: Allows teachers to view the list of subjects assigned to them.
* **Assign Grades**: Allows teachers to assign grades to students for a specific subject.
* **Update Grades**: Allows teachers to update the grades of students.
* **View Grades**: Allows teachers to view the grades of students.
* **Attendance for a Class**: Allows teachers to mark attendance for a class.
* **Update Attendance**: Allows teachers to update the attendance of students.

## Student Functionalities

* **View Attendance**: Allows students to view their attendance records.
* **View Grades**: Allows students to view their grades.
* **View Percentage**: Allows students to view their overall percentage based on grades.
* **View Subjects Assigned**: Allows students to view the subjects assigned to them.