SHAHEED ZULFIKAR ALI BHUTTO
INSTITUTE OF SCIENCE & TECHNOLOGY

SOFTWARE DESIGN AND ARCHITECTURE

STUDENT MANAGEMENT SYSTEM

PREPARED BY :

1. M. ARBAAZ HUSSAIN (2380197)
2. SYED HUSSAIN (2380219)
3. ASIM RAZA (2380183)

COURSE INSTRUCTOR:

AMIR IMAM

DATE: 5/29/2025

🖹 Student Management System – Documentation

1. Introduction

1.1 Purpose

The Student Management System is designed to manage information related to students, including personal details, academic records, and course enrollments. This application facilitates administrative tasks within an academic institution.

1.2 Scope

This software provides a simple Windows Forms-based interface for adding, editing, and deleting student records. It may also include functionality for managing courses, grades, and departments (to be validated from deeper inspection).

1.3 Intended Audience

- Academic institutions
- School administrators
- Software development teams
- QA and testers

1.4 Definitions and Acronyms

- **SRS**: Software Requirements Specification
- **WinForms**: Windows Forms, a UI framework for building Windows desktop applications □ **CRUD**: Create, Read, Update, Delete

2. Overall Description

2.1 Product Perspective

This is a standalone desktop application. It uses a local database (likely via SQL Server or file-based DB) and has a modular structure with forms representing various functions (e.g., student details, course management).

2.2 Product Functions

- Add new student records
- Edit or delete existing student data
- View all students in a grid or list
- Possibly manage courses, grades, and class schedules

2.3 User Characteristics

Users are expected to have basic computer skills. No programming knowledge is required to operate the system.

2.4 Constraints

- Windows-only platform
- Requires .NET Framework compatible with Windows Forms
- May require a local database setup

2.5 Assumptions and Dependencies

- Users will have administrative rights to install and run the application
- Data will be stored and retrieved from a local database

3. Specific Requirements

3.1 Functional Requirements

- **FR1**: The system shall allow the user to add a new student with name, ID, contact details, and course info.
- **FR2**: The system shall validate user input to ensure data integrity.
- **FR3**: The system shall allow updating and deleting student records.
- **FR4**: The system shall display a list of all students in a tabular format.
- **FR5**: The system shall provide search functionality to filter student records.

3.2 Non-Functional Requirements

- **NFR1**: The system should respond to user actions within 1 second.
- **NFR2**: The UI should be intuitive and easy to navigate.
- **NFR3**: Data should be stored securely using access restrictions.

3.3 Interface Requirements

- **User Interface**: Windows Forms GUI with buttons, textboxes, grids
- **Database Interface**: SQL-based queries to interact with a local database

📂 Project Structure

```
SDA project/
│
├── hosts.txt
├── prompts.txt
├── StudentManagmentSystem/
│   ├── SchoolAPI/                 # ASP.NET Core Web API Backend
│   │   ├── Controllers/           # API endpoints
│   │   ├── Models/                # Data models (Student, Course, etc.)
│   │   ├── Migrations/            # EF Core migrations
│   │   ├── Program.cs             # Entry point of the API
│   │   ├── appsettings.json       # Configuration files
│   │   └── SchoolAPI.csproj       # Project file
│   └── StudentManagement.sln     # Solution file │
├── UI/
│   ├── CourseService/             # Handles course-related service logic
│   ├── StudentCourseService/      # Manages student-course relations
│   └── StudentManagementUI/
│       ├── StudentManagementUI/   # Windows Forms App
│       └── StudentManagementUI.sln # UI solution file
│
```

✓ Features

Backend (ASP.NET Core Web API)

- **Controllers** to manage:
    - Students ○ Courses
    - Student-Course Enrollment
- **EF Core Migrations** for database schema evolution
- **appsettings.json** for configuration

Frontend (Windows Forms)

- Windows Forms UI built in C#
- Divided into services for **Courses**, **Student-Course**, and **Main UI**
- Uses a service-oriented architecture to interact with the backend

⚙ How to Run the Project

🖥 Backend (API)

1. Open `StudentManagement.sln` in Visual Studio.

2. Make sure `SchoolAPI` is set as the startup project.
3. Run the project (F5) — the API will start on the configured port.
4. Check `appsettings.json` for port and DB configuration.

🖼 Frontend (Windows Forms UI)

1. Open `StudentManagementUI.sln` in Visual Studio.
2. Set `StudentManagementUI` as the startup project.
3. Run the UI (F5) — it will attempt to connect to the backend API.

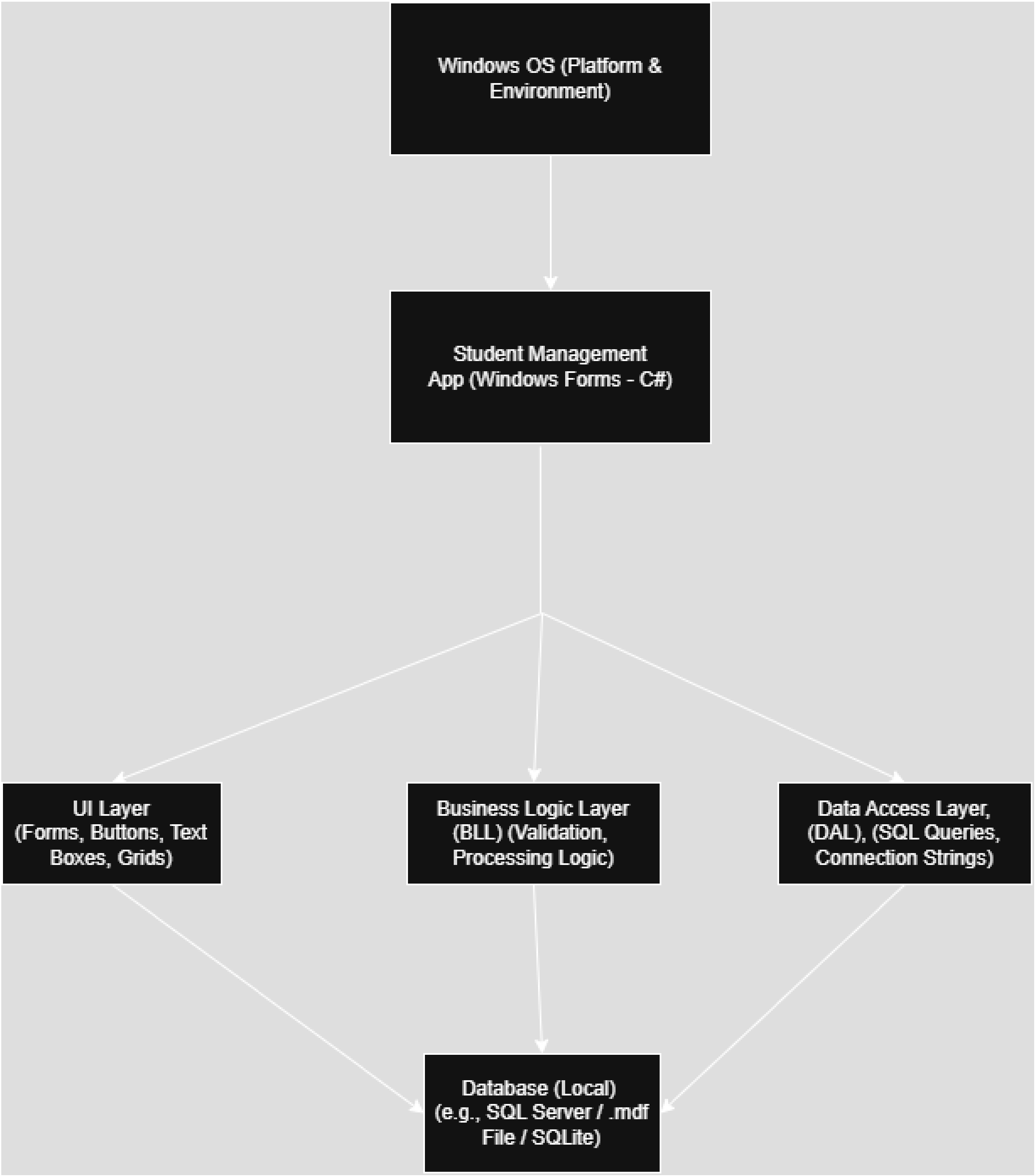
4.

🔧 Ensure both projects run simultaneously and the UI targets the correct API URL.

📄 Configuration Files

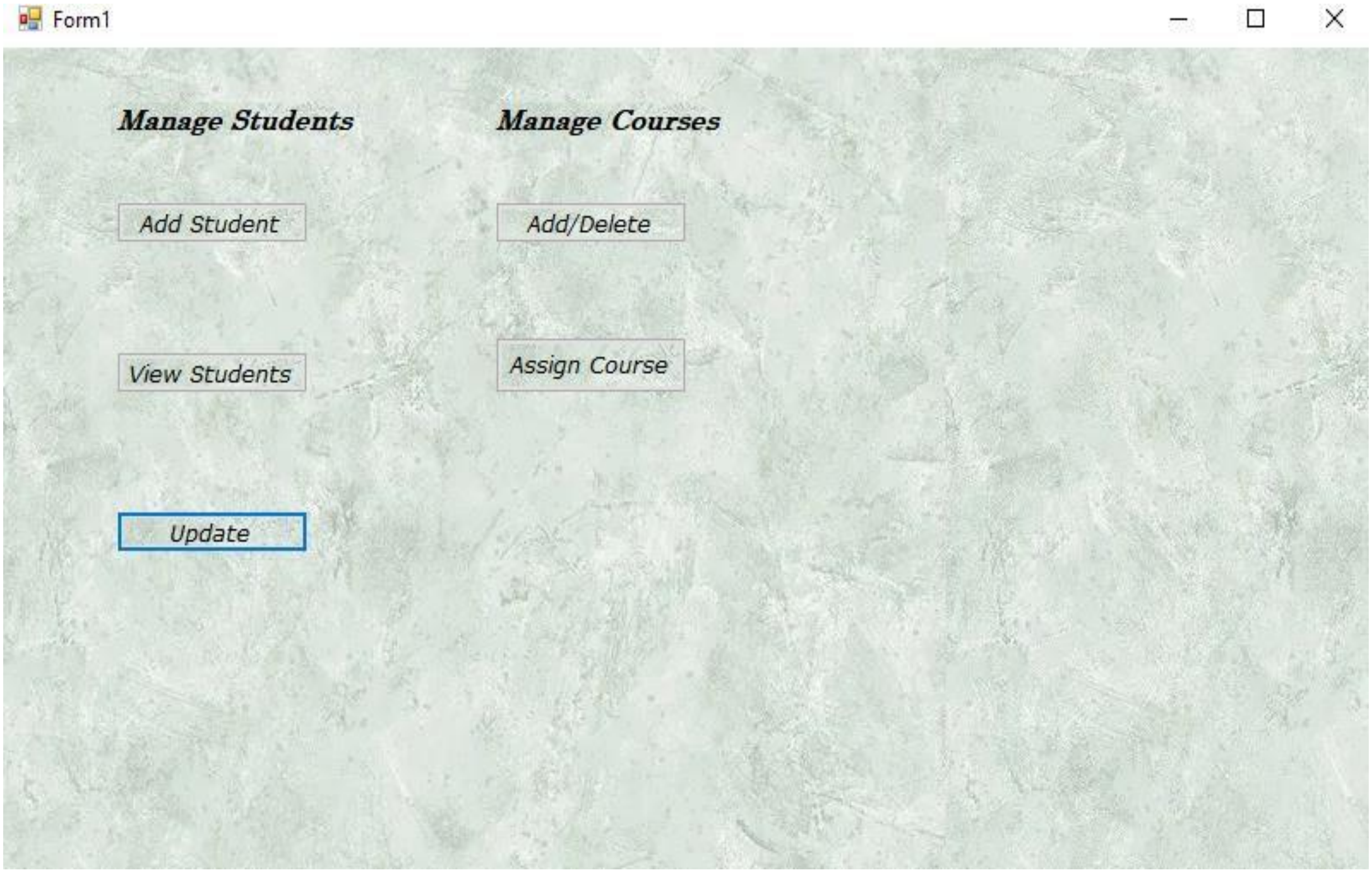
- `appsettings.json`
  Contains connection strings and logging configuration.
- `SchoolAPI.http`
  HTTP request templates for testing endpoints (Visual Studio feature).

ARCHITECTURE DIAGRAM

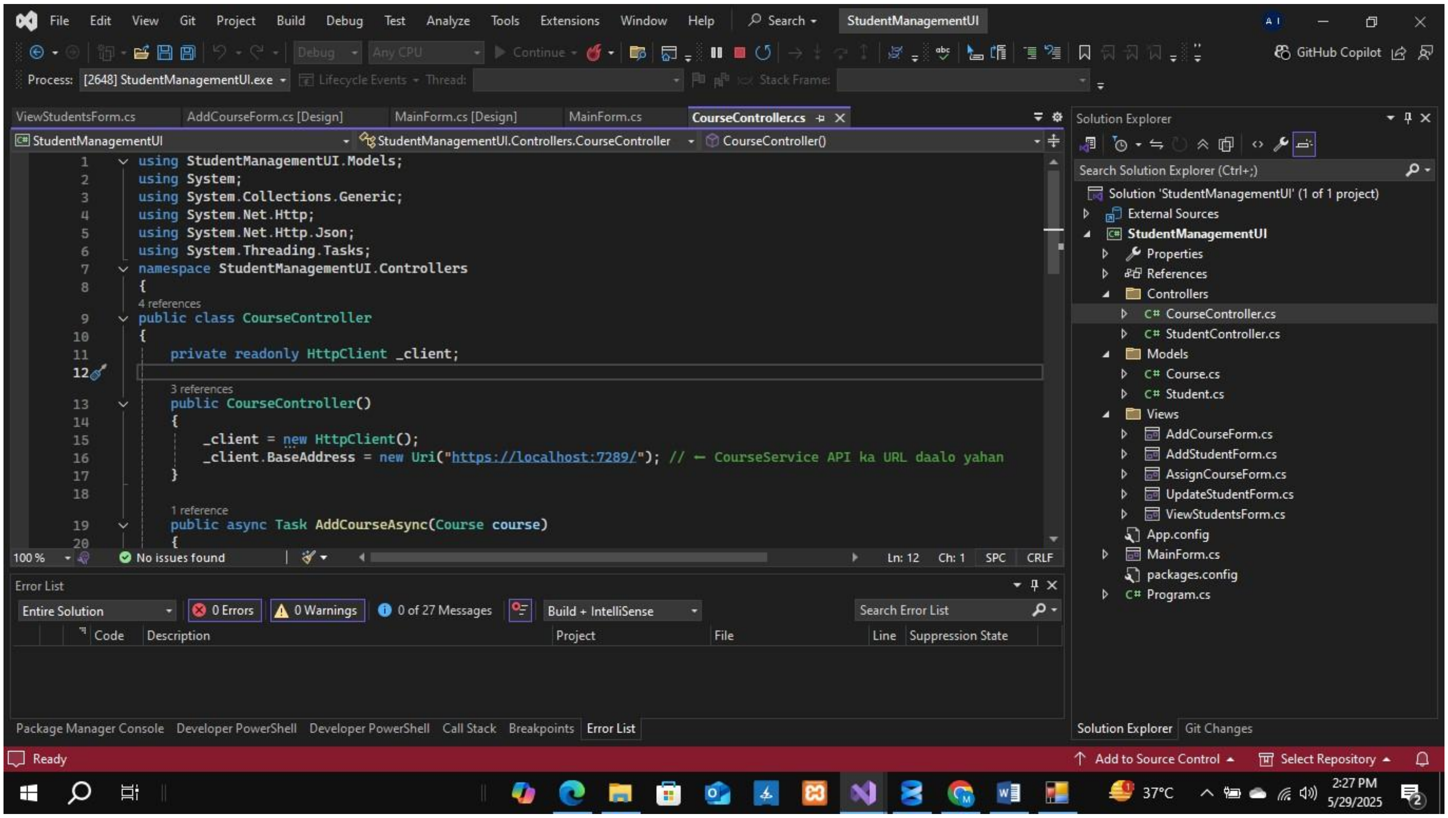

Windows OS (Platform & Environment)
Student Management App (Windows Forms - C#)
UI Layer (Forms, Buttons, Text Boxes, Grids)
Business Logic Layer (BLL) (Validation, Processing Logic)
Data Access Layer, (DAL), (SQL Queries, Connection Strings)
Database (Local) (e.g., SQL Server / .mdf File / SQLite)

Explanation of Layers

- **UI Layer**: Handles user input and output (Windows Forms).
- **Business Logic Layer (BLL)**: Validates data, applies rules (e.g., "no duplicate student IDs").
- **Data Access Layer (DAL)**: Contains code to interact with the database.
- **Database**: Stores student data (e.g., names, IDs, courses).



COURSE CONTROLLER:

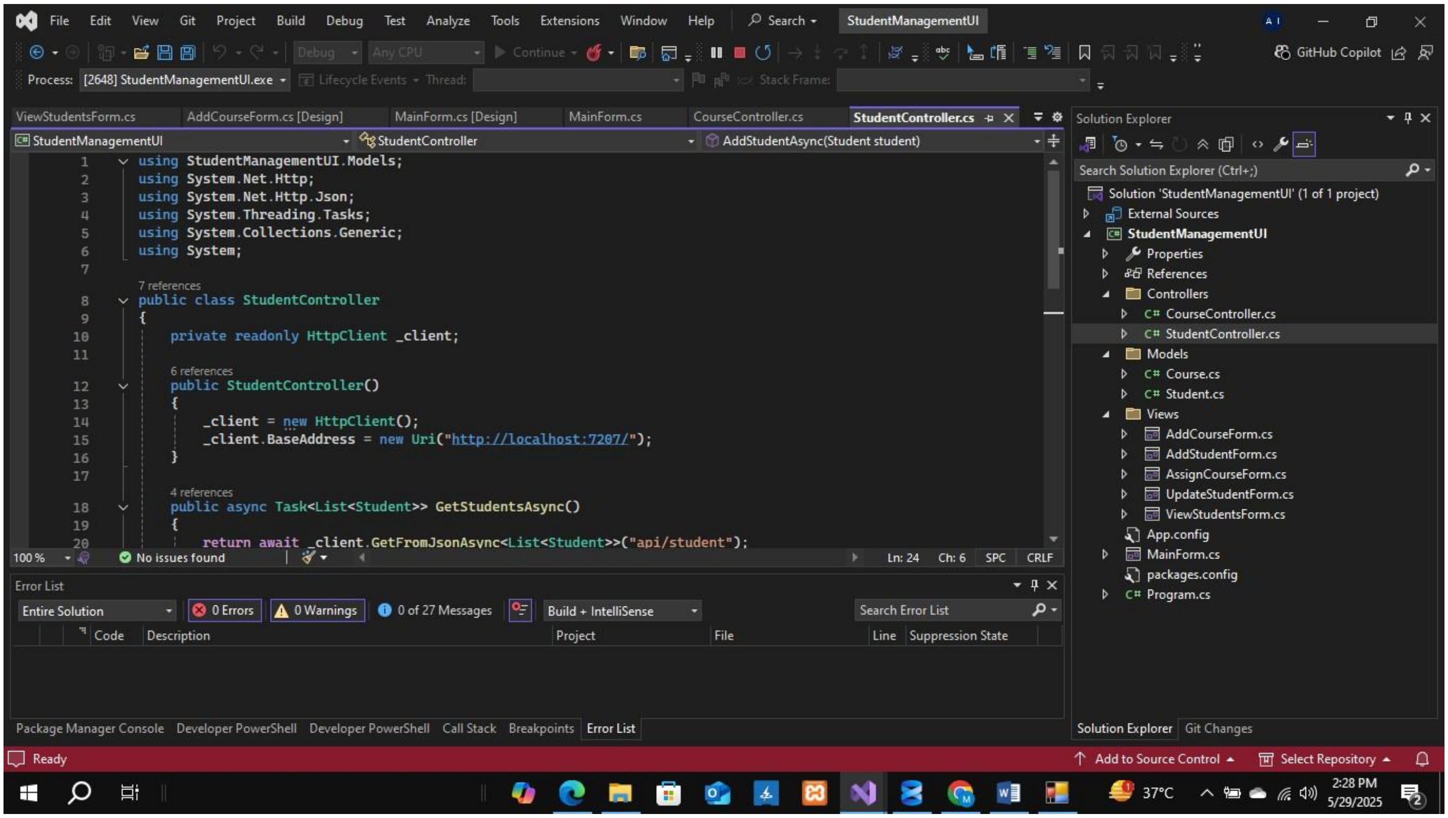


```
using StudentManagementUI.Models;
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
namespace StudentManagementUI.Controllers
{
4 references
public class CourseController
{
    private readonly HttpClient _client;

    3 references
    public CourseController()
    {
        _client = new HttpClient();
        _client.BaseAddress = new Uri("https://localhost:7289/"); // ← CourseService API ka URL daalo yahan
    }

    1 reference
    public async Task AddCourseAsync(Course course)
    {
```

STUDENTCONTROLLER.CS:



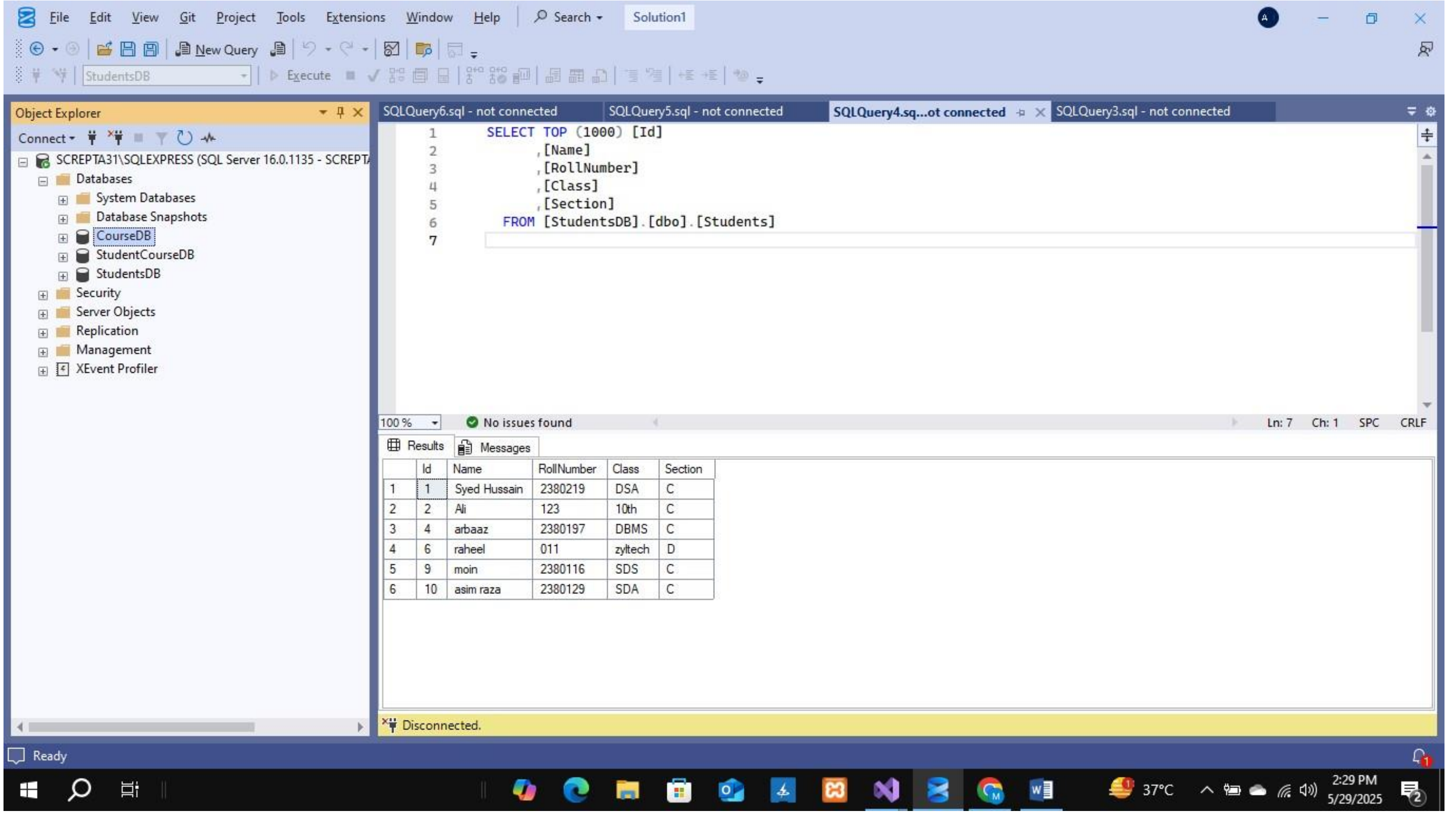
```
using StudentManagementUI.Models;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using System.Collections.Generic;
using System;

7 references
public class StudentController
{
    private readonly HttpClient _client;

    6 references
    public StudentController()
    {
        _client = new HttpClient();
        _client.BaseAddress = new Uri("http://localhost:7207/");
    }

    4 references
    public async Task<List<Student>> GetStudentsAsync()
    {
        return await _client.GetFromJsonAsync<List<Student>>("api/student");
```

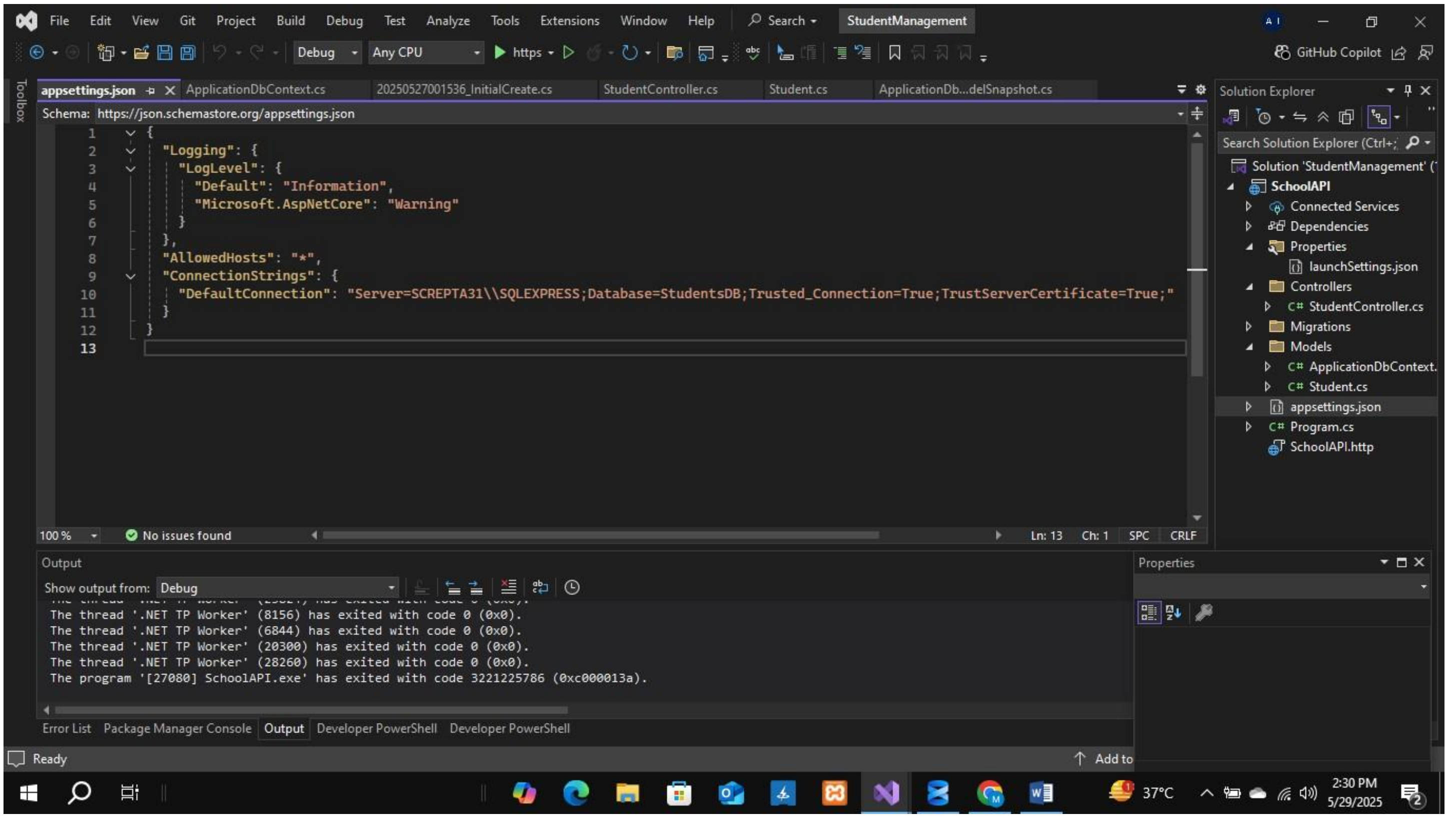
DATABASE:



```sql
SELECT TOP (1000) [Id]
      ,[Name]
      ,[RollNumber]
      ,[Class]
      ,[Section]
  FROM [StudentsDB].[dbo].[Students]
```

	Id	Name	RollNumber	Class	Section
1	1	Syed Hussain	2380219	DSA	C
2	2	Ali	123	10th	C
3	4	arbaaz	2380197	DBMS	C
4	6	raheel	011	zytech	D
5	9	moin	2380116	SDS	C
6	10	asim raza	2380129	SDA	C

CONNECTION WITH DB:

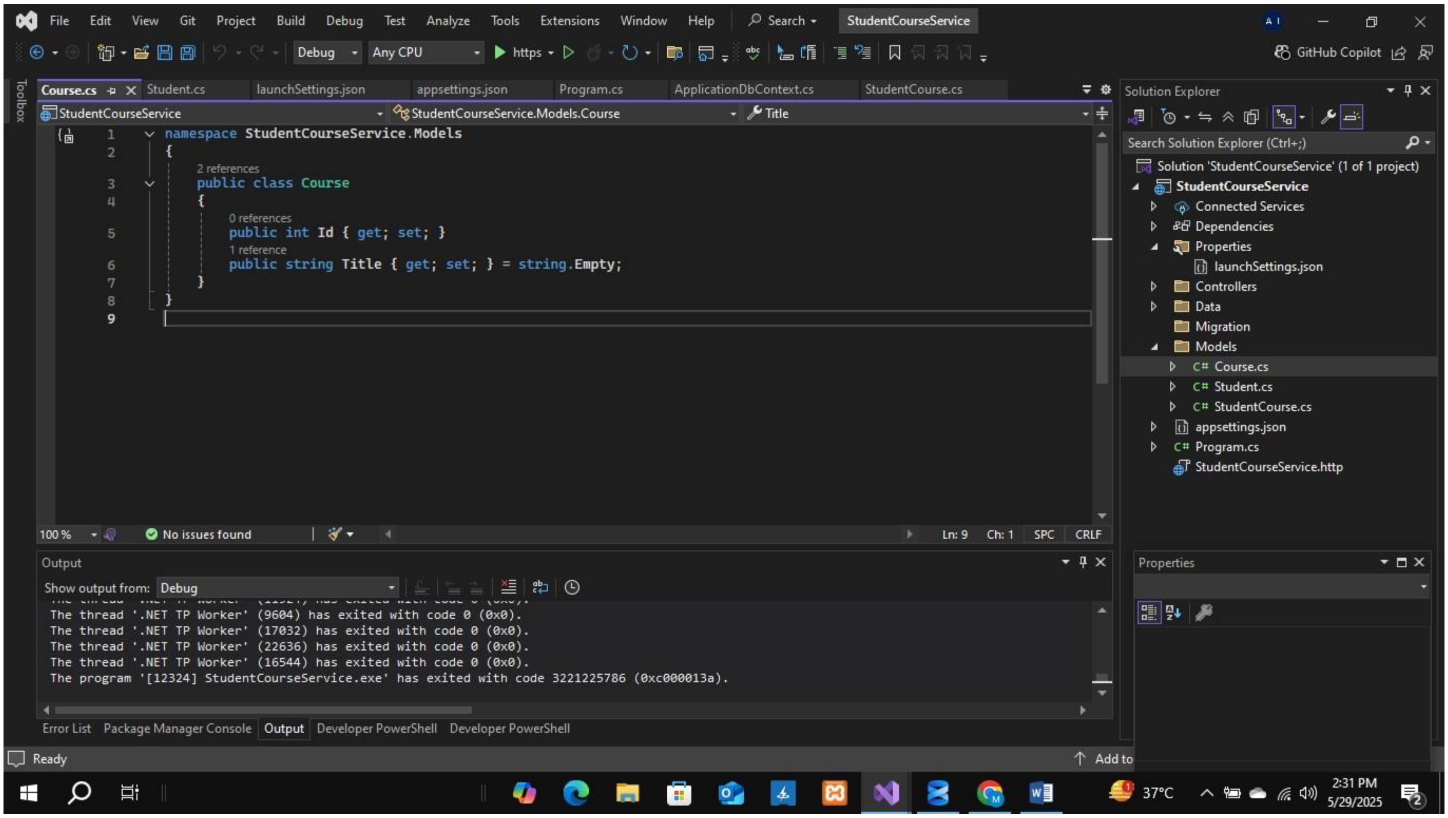


```json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=SCREPTA31\\SQLEXPRESS;Database=StudentsDB;Trusted_Connection=True;TrustServerCertificate=True;"
  }
}
```

```
The thread '.NET TP Worker' (8156) has exited with code 0 (0x0).
The thread '.NET TP Worker' (6844) has exited with code 0 (0x0).
The thread '.NET TP Worker' (20300) has exited with code 0 (0x0).
The thread '.NET TP Worker' (28260) has exited with code 0 (0x0).
The program '[27080] SchoolAPI.exe' has exited with code 3221225786 (0xc000013a).
```

MODELS:

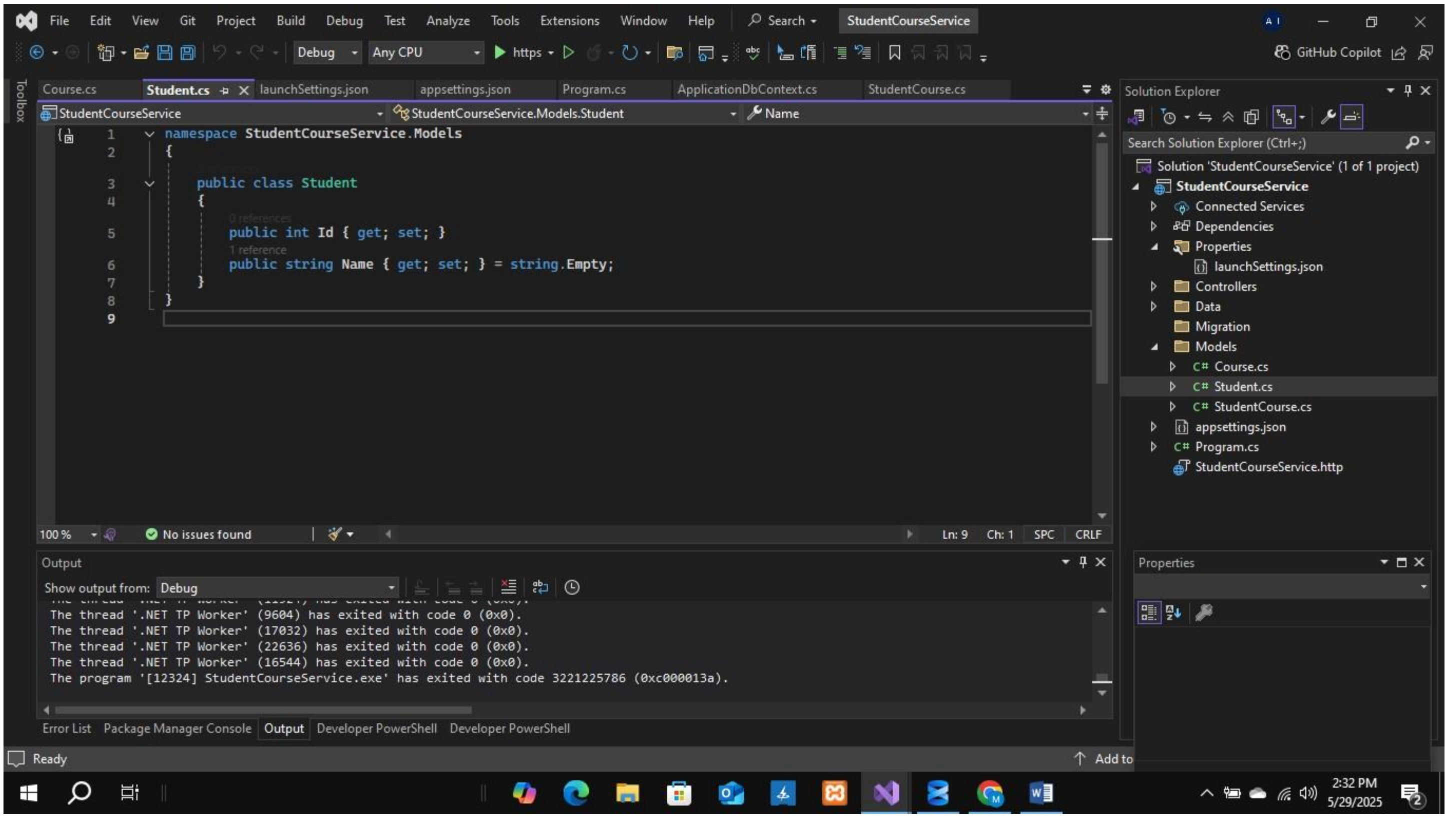
COURSE.CS



```csharp
namespace StudentCourseService.Models
{
    2 references
    public class Course
    {
        0 references
        public int Id { get; set; }
        1 reference
        public string Title { get; set; } = string.Empty;
    }
}
```

```
The thread '.NET TP Worker' (9604) has exited with code 0 (0x0).
The thread '.NET TP Worker' (17032) has exited with code 0 (0x0).
The thread '.NET TP Worker' (22636) has exited with code 0 (0x0).
The thread '.NET TP Worker' (16544) has exited with code 0 (0x0).
The program '[12324] StudentCourseService.exe' has exited with code 3221225786 (0xc000013a).
```

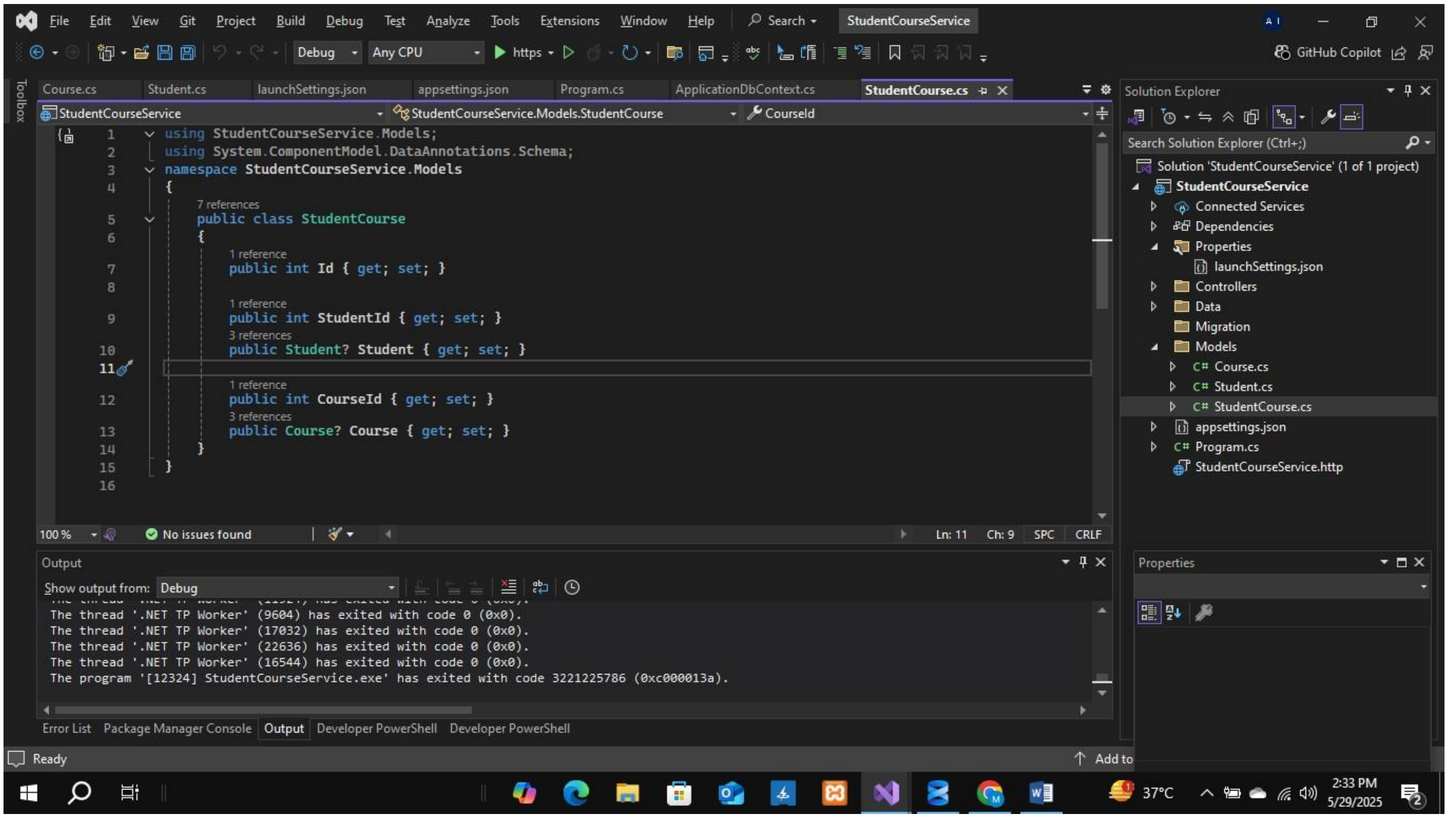
STUDENT.CS:



```csharp
namespace StudentCourseService.Models
{
    public class Student
    {
        0 references
        public int Id { get; set; }
        1 reference
        public string Name { get; set; } = string.Empty;
    }
}
```

```
The thread '.NET TP Worker' (9604) has exited with code 0 (0x0).
The thread '.NET TP Worker' (17032) has exited with code 0 (0x0).
The thread '.NET TP Worker' (22636) has exited with code 0 (0x0).
The thread '.NET TP Worker' (16544) has exited with code 0 (0x0).
The program '[12324] StudentCourseService.exe' has exited with code 3221225786 (0xc000013a).
```

STUDENTCOURSE.CS:



```csharp
using StudentCourseService.Models;
using System.ComponentModel.DataAnnotations.Schema;
namespace StudentCourseService.Models
{
    public class StudentCourse
    {
        public int Id { get; set; }

        public int StudentId { get; set; }
        public Student? Student { get; set; }

        public int CourseId { get; set; }
        public Course? Course { get; set; }
    }
}
```

```
The thread '.NET TP Worker' (9604) has exited with code 0 (0x0).
The thread '.NET TP Worker' (17032) has exited with code 0 (0x0).
The thread '.NET TP Worker' (22636) has exited with code 0 (0x0).
The thread '.NET TP Worker' (16544) has exited with code 0 (0x0).
The program '[12324] StudentCourseService.exe' has exited with code 3221225786 (0xc000013a).
```