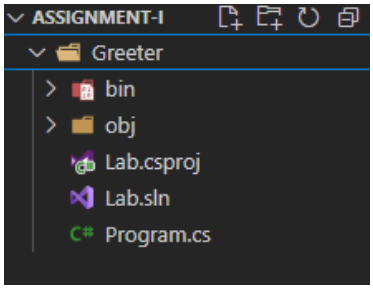
Assignment 1 (Starter)

1. **Create a new console application named Greeter under Assignment 1 folder**



2. **Modify Program.cs to define a variable fullName and assign some name.**

Code:

Greeter > C# Program.cs

```
string fullName = "Ajay Pradhan";
Console.WriteLine(fullName);
```

3. Print value of fullName to console

Output:

```
Ajay Pradhan
```

4. Define another variable cFullName and initialize it with fullName in uppercase letters. (Hint: ToUpper() string helper)

Code:

```
Console.WriteLine(fullName);
string cfullName = fullName.ToUpper();
Console.WriteLine("Hello, " +cfullName + "!");
```

5. Print value of cFullName to console in format: "Hello, BISHNU RAWAL!".

Output:

```
Hello, Ajay Pradhan!
```

6. Instead of initilizing fullName, get it from user.

Code:

```
Console.WriteLine("What is your name?");
string? name = Console.ReadLine();
Console.WriteLine("Name: " + name);
```

Output:

```
What is your name?
Ajay Pradhan
```

7. Now also ask user to enter his/her "Date of Birth" and display user friendly date to console. And 8. Your last task is to calculate his/her age as accurate as possible and to display its console.

Code:

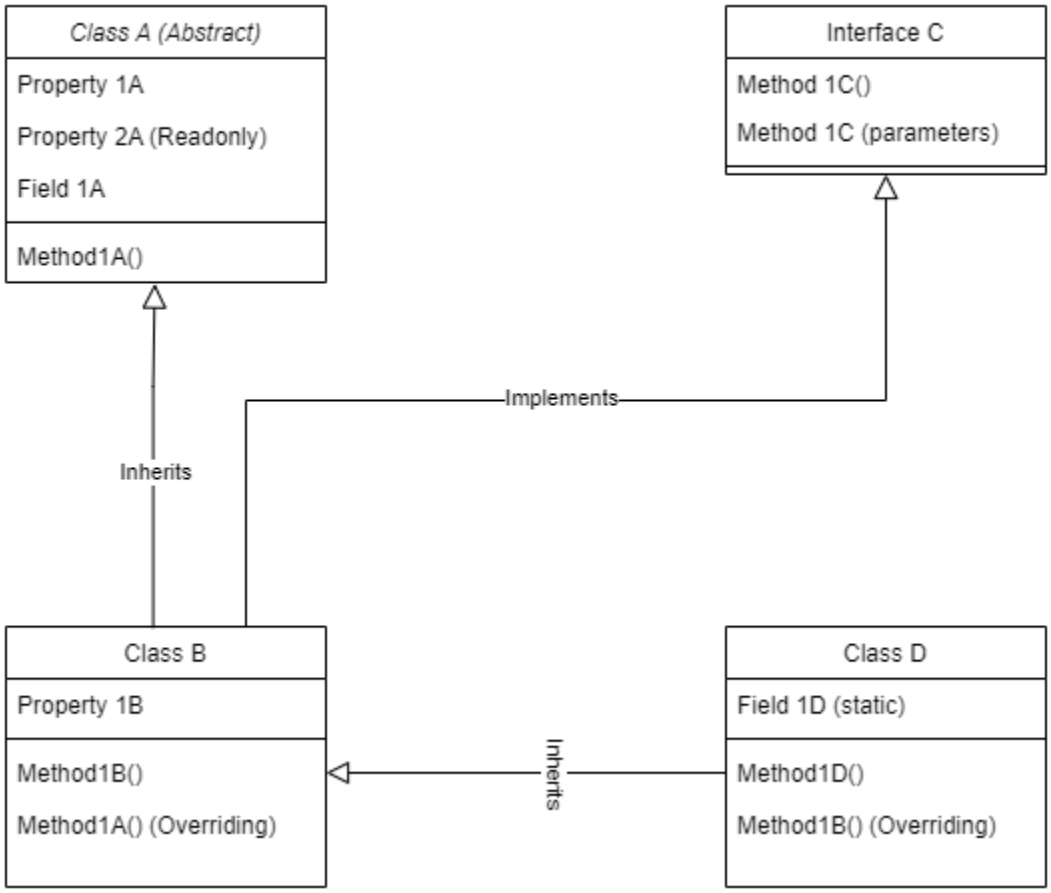
```
Console.WriteLine("What is your name?");
string? name = Console.ReadLine();
// Console.WriteLine("Name: " + name);

Console.WriteLine("Enter your Dob:(MM/DD/YYYY) ");
string? DobInput = Console.ReadLine();
DateTime Dob = DateTime.Parse(DobInput);

Console.WriteLine("Hello, " + name + "!");
Console.WriteLine("Your DOB: " + Dob.ToString("dddd,dd MMMM yyyy"));

string GetAge(DateTime dob)
{
    TimeSpan age = DateTime.Now - dob;
    var ageInDay = age.TotalDays;
    var Years = (int)(ageInDay / 365);
    var RemDays = ageInDay % 365;
    var months = (int)(RemDays / 30);
    var daysAfterMonths = (int)(RemDays % 30);
    var weeks = daysAfterMonths % 7;
    int years = age.Days / 365;
    int months = (age.Days % 365) / 30;
    int weeks = ((age.Days % 365) % 30) / 7;
    int days = ((age.Days % 365) % 30) % 7;
    return $"{Years} Year {months} Months {weeks} Week {daysAfterMonths} Days";
}
string? age = GetAge(Dob);
Console.WriteLine("Your Age: " + age);
```

Output:

```
PS E:\Bsc.CSIT\6th semester\Net Centric Computing\Lab> dotnet run
What is your name?
Ajay Pradhan
Enter your Dob:(MM/DD/YYYY)
02/13/2001
Hello, Ajay Pradhan!
Your DOB: Tuesday,13 February 2001
Your Age: 23 Year 4 Months 1 Week 29 Days
PS E:\Bsc.CSIT\6th semester\Net Centric Computing\Lab>
```

Assignment-2(OOP CONCEPT)

Think of a real world scenario where you can design class(s), interface(s) and members as shown in following class diagram:



Description

Class A (Abstract): Vehicle

The 'Vehicle' class is an abstract class that defined common properties and methods for various types of vehicles.

Code:

C# Car.cs × C# Program.cs

C# Car.cs > ElectricCar

```
3 references
public abstract class Vehicle
{
    2 references
    public int VIN {get;}
    2 references
    public string Brand { get; set;}
    2 references
    string model = string.Empty;
    2 references
    public string Model
    {
        get { return model; }
        set {model = value; }
    }
    1 reference
    public Vehicle(int vin,string brand ,string model)
    {
        VIN = vin;
        Brand = brand;
        Model = model;
    }
    6 references
    public virtual string DisplayInfo()
    {
        var vehicleInfo = $"Brand name:{Brand}\nModel:{Model}\nVIN:{VIN}";
        return vehicleInfo;
    }
}
```

Class B: Car

The Car class is a derived class of the Vehicle abstract class and implements the IFuelEfficient interface.

Code:

Car.cs > ElectricCar

```
4 references
class Car : Vehicle, IFuelEfficient
{
    2 references
    public int ProducedYear {get;set;}
    3 references
    private int fuelEfficiency;

    2 references
    public Car(int vin, string brand, string model, int year):base(vin,brand, model)
    {
        ProducedYear = year;
    }
    1 reference
    public int DisplayFuelEfficiency()=> fuelEfficiency;

    2 references
    public void SetFuelEfficiency(int efficiency)
    {
        fuelEfficiency = efficiency;
    }
    6 references
    public override string DisplayInfo()
    {
        var details =  base.DisplayInfo();
        details += $"\nReleased Year:{ProducedYear}\nFuel Efficiency:{fuelEfficiency} Km/l";
        return details;
    }

}
```

Class D: Ecar

The ElectricCar class is derived from the Car class and has a static (Static fields are often used to store values that are common to all instances of a class) property BatteryCapacity.

Code:

```
C# Car.cs   ×   C# Program.cs
C# Car.cs > ElectricCar
        2 references
54      class ElectricCar:Car
55      {
            2 references
56          public static int BatteryCapacity;
            1 reference
57          public ElectricCar(int vin,string brand, string model, int year, int range)
58          :base(vin, brand,model,year)
59          {
60              BatteryCapacity = range;
61          }
            0 references
62          public void ChargeCar()
63          {
64              Console.WriteLine("Car is charging");
65          }
            6 references
66          public override string DisplayInfo()
67          {
68              var details = base.DisplayInfo();
69              details += $"\nBattery capacity: {BatteryCapacity} kwh";
70              return details;
71          }
72      }
```

Interface C

The IFuelEfficient interface is a blueprint that defines a single method called DisplayFuelEfficiency().

Code:

```
C# Car.cs   ×   C# Program.cs
C# Car.cs > ElectricCar
        1 reference
48      interface IFuelEfficient
49      {
            1 reference
50          public int DisplayFuelEfficiency();
            2 references
51          void SetFuelEfficiency(int efficiency);
52      }
```

Program.cs

C# Program.cs

C# Program.cs

```
var car = new Car(1234,"Nissan","X-TRAIL",2018);
car.SetFuelEfficiency(38);
Console.WriteLine(car.DisplayInfo());

Console.WriteLine("----------------------------------");

var ecar = new ElectricCar(5678,"MG"," MG4 EV",2023, 450);
Console.WriteLine(ecar.DisplayInfo());
```

Output:

```
PS E:\Bsc.CSIT\6th semester\Net Centric Computing\Lab\Assignment-II> dotnet run
Brand name:Nissan
Model:X-TRAIL
VIN:1234
Released Year:2018
Fuel Efficiency:38 Km/l
----------------------------------
Brand name:MG
Model: MG4 EV
VIN:5678
Released Year:2023
Fuel Efficiency:0 Km/l
Battery capacity: 450 kwh
PS E:\Bsc.CSIT\6th semester\Net Centric Computing\Lab\Assignment-II>
```

Assignment 3 (File Handling and LINQ)

You have csv data for inflation rate in Asia and the Pacific:

RegionalMember	Year	Inflation	Unit of Measurement	Subregion	Country Code
.....	.....	....	....	.....	.....
Armenia	2022	8.6	%	Central Asia	ARM
Armenia	2023	7	%	Central Asia	ARM
Armenia	2024	6.2	%	Central Asia	ARM
Azerbaijan	2018	2.4	%	Central Asia	AZE
Azerbaijan	2019	2.7	%	Central Asia	AZE
Azerbaijan	2020	2.8	%	Central Asia	AZE
.....	.....	....	....	.....	.....

1. Create class **Inflation** with all column headers in csv file as properties.

Code:

C# Inflation.cs > ...

```
7 references
public class Inflation
{
    6 references
    public required string RegionalMember { get; set; }
    5 references
    public int Year { get; set; }
    7 references
    public double? InflationRate { get; set; }
    1 reference
    public required string UnitOfMeasurement { get; set; }
    1 reference
    public required string Subregion { get; set; }
    1 reference
    public required string CountryCode { get; set; }
}
```

2. Create another class **InflationAnalysis** with methods as needed to

i. Read csv text file and populate `List<Inflation>` collection with the data read.

Code:

C# InflationAnalysis.cs > InflationAnalysis > LoadData

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using CsvHelper;
using CsvHelper.Configuration;

1 reference
public class InflationAnalysis
{
    5 references
    public List<Inflation> Inflations { get; set; } = new List<Inflation>();

    1 reference
    public void LoadData(string filePath)
    {
        using (var reader = new StreamReader(filePath))
        using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))
        {
            var config = new CsvConfiguration(CultureInfo.InvariantCulture)
            {
                PrepareHeaderForMatch = args => args.Header.ToLower(),
            };

            csv.Context.RegisterClassMap<InflationMap>();
            Inflations = csv.GetRecords<Inflation>().ToList();
        }
    }
```

ii. To answer following queries related to inflation

a. Find inflation rates for countries for the year 2021.

Code:

```
1 reference
public List<Inflation> GetInflationRatesForYear(int year)
{
    return Inflations.Where(i => i.Year == year).ToList();
}
```

Output:

```
Inflation rates for 2021:
Developing Asia: 2.6%
Developing Asia excluding the PRC: 4.2%
Caucasus and Central Asia: 9%
Armenia: 7.2%
Azerbaijan: 6.7%
Georgia: 9.6%
Kazakhstan: 8%
Kyrgyz Republic: 11.9%
Tajikistan: 8%
Turkmenistan: 12.5%
Uzbekistan: 10.7%
East Asia: 1.1%
Hong Kong, China: 1.6%
Mongolia: 7.3%
People's Republic of China: 0.9%
Republic of Korea: 2.5%
Taipei,China: 2%
South Asia: 5.8%
Afghanistan: 5.2%
Bangladesh: 5.6%
Bhutan: 7.3%
India: 5.5%
Maldives: 0.5%
Nepal: 3.6%
Pakistan: 8.9%
Sri Lanka: 6%
Southeast Asia: 2%
Brunei Darussalam: 1.7%
Cambodia: 2.9%
Indonesia: 1.6%
Lao People?s Dem. Rep.: 3.8%
Malaysia: 2.5%
Myanmar: 3.6%
Philippines: 3.9%
Singapore: 2.3%
Thailand: 1.2%
Timor-Leste: 3.8%
Viet Nam: 1.8%
The Pacific: 3.1%
Cook Islands: 1.8%
Federated States of Micronesia: 1.8%
Fiji: 0.2%
Kiribati: 1%
Marshall Islands: 2.2%
Nauru: 1.2%
Niue: %
Palau: 0.5%
Papua New Guinea: 4.5%
Samoa: -3%
Solomon Islands: -0.2%
Tonga: 1.4%
Tuvalu: 6.7%
Vanuatu: 2.3%
```

b. A year when Nepal has highest inflation.

Code:

```
1 reference
public int GetYearWithHighestInflationForCountry(string country)
{
    return Inflations
        .Where(i => i.RegionalMember.Equals(country, StringComparison.OrdinalIgnoreCase))
        .OrderByDescending(i => i.InflationRate)
        .FirstOrDefault()?.Year ?? 0;
}
```

Output:

```
Year when Nepal had the highest inflation: 2023
```

c. List top 10 regions (countries) where inflation is highest for all time.

Code:

```
1 reference
public List<Inflation> GetTopRegionsWithHighestInflation(int topCount)
{
    return Inflations
        .OrderByDescending(i => i.InflationRate)
        .Take(topCount)
        .ToList();
}
```

Output:

```
Top 10 regions with the highest inflation:
Sri Lanka: 46.4% in 2022
Pakistan: 27.5% in 2023
Sri Lanka: 24.6% in 2023
Lao People?s Dem. Rep.: 23% in 2022
Uzbekistan: 17.5% in 2018
Lao People?s Dem. Rep.: 16% in 2023
Myanmar: 16% in 2022
Mongolia: 15.2% in 2022
Mongolia: 15.2% in 2022
Mongolia: 15.2% in 2022
Kazakhstan: 15% in 2022
Pakistan: 15% in 2024
```

d. List top 3 south asian countries with lowest inflation rate for year 2020

Code:

```
1 reference
public List<Inflation> GetLowestInflationRatesForYear(int year, int topCount, List<string> countries)
{
    return Inflations
        .Where(i => i.Year == year && countries.Contains(i.RegionalMember))
        .OrderBy(i => i.InflationRate)
        .Take(topCount)
        .ToList();
}
}
```

Output:

```
Top 3 South Asian countries with lowest inflation rate in 2020:
Maldives: -1.4%
Sri Lanka: 4.6%
Afghanistan: 5.6%
```

InflationMap.cs

It is used to map the columns in the CSV file to the properties of the 'Inflation' class ensuring that the data from each column in the CSV file is correctly assigned.

Code:

```
2 references
public sealed class InflationMap : ClassMap<Inflation>
{
    0 references
    public InflationMap()
    {
        Map(m => m.RegionalMember).Name("RegionalMember");
        Map(m => m.Year).Name("Year");
        Map(m => m.InflationRate).Name("Inflation");
        Map(m => m.UnitOfMeasurement).Name("Unit of Measurement");
        Map(m => m.Subregion).Name(" Subregion");
        Map(m => m.CountryCode).Name(" Country Code");
    }
}
```

Program.cs

C# Program.cs > ...

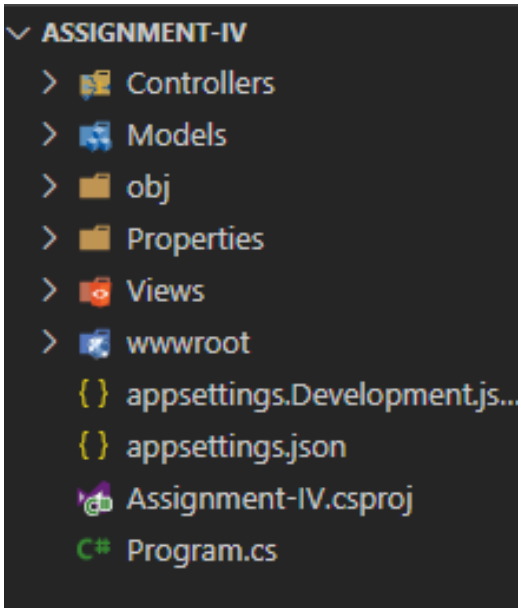
```
using System;
using System.Collections.Generic;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        var analysis = new InflationAnalysis();
        string filePath = "Inflation.csv";
        analysis.LoadData(filePath);

        var inflation2021 = analysis.GetInflationRatesForYear(2021);
        Console.WriteLine("Inflation rates for 2021:");
        foreach (var item in inflation2021)
        {
            Console.WriteLine($"{item.RegionalMember}: {item.InflationRate}%");
        }

        int highestInflationYearForNepal = analysis.GetYearWithHighestInflationForCountry("Nepal");
        Console.WriteLine($"Year when Nepal had the highest inflation: {highestInflationYearForNepal}");

        var topRegions = analysis.GetTopRegionsWithHighestInflation(10);
        Console.WriteLine("Top 10 regions with the highest inflation:");
        foreach (var item in topRegions)
        {
            Console.WriteLine($"{item.RegionalMember}: {item.InflationRate}% in {item.Year}");
        }

        var southAsianCountries = new List<string> { "Afghanistan", "Bangladesh", "Bhutan", "India",
        "Maldives", "Nepal", "Pakistan", "Sri Lanka" };
        var lowestInflation2020 = analysis.GetLowestInflationRatesForYear(2020, 3, southAsianCountries);
        Console.WriteLine("Top 3 South Asian countries with lowest inflation rate in 2020:");
        foreach (var item in lowestInflation2020)
        {
            Console.WriteLine($"{item.RegionalMember}: {item.InflationRate}%");
        }
    }
}
```

Assignment 4 (ASP .NET Core MVC)

Student Management System

Task 1: Setting up the Project

1. **Create a new ASP.NET Core MVC project.**



2. Install necessary NuGet packages for SQLite (Microsoft.EntityFrameworkCore.Sqlite) and API calls (System.Net.Http.Json).

```
dotnet add package Microsoft.Data.Sqlite
```

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

3. **Set up SQLite database context for student management.**

Data > C# StudentDbContext.cs > ...

```
using Microsoft.EntityFrameworkCore;
using Ajay.Models;

namespace Ajay.Data
{
    2 references
    public class StudentDbContext : DbContext
    {
        0 references
        public StudentDbContext(DbContextOptions<StudentDbContext> options) : base(options)
        {
        }

        0 references
        public DbSet<Student> Students { get; set; }

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Student>().ToTable("Students");
        }
    }
}
```

Task 2: Create Models

1. Create a Student model with properties like Id, Name, Email, and Age.

```
public class Students
{
    0 references
    public int Id { get; set; }
    0 references
    public string Name { get; set; }
    0 references
    public string Address { get; set; }
    0 references
    public char Gender { get; set; }
    0 references
    public string Email { get; set; }
    0 references
    public DateTime DateOfBirth { get; set; }
}
```

3. **Add data annotations for field validations (e.g., Required, EmailAddress, Range).**

```
using System.ComponentModel.DataAnnotations;

0 references
public class RegisterViewModel
{
    [Required]
    0 references
    public string Username { get; set; }

    [Required]
    [DataType(DataType.Password)]
    0 references
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    0 references
    public string ConfirmPassword { get; set; }
}
```

Task 3: Create Views

1. **Create views for listing all students, creating a new student, editing a student, and deleting a student.**

```
@model LoginViewModel

<h2>Login</h2>

<form asp-action="Login">
    <div>
        <label asp-for="Username"></label>
        <input asp-for="Username" />
        <span asp-validation-for="Username"></span>
    </div>
    <div>
        <label asp-for="Password"></label>
        <input asp-for="Password" type="password" />
        <span asp-validation-for="Password"></span>
    </div>
    <div>
        <input asp-for="RememberMe" type="checkbox" />
        <label asp-for="RememberMe"></label>
    </div>
    <a asp-controller="Account" asp-action="Register">Register</a>
    <button type="submit">Login</button>
</form>

@if (ViewBag.Error != null)
{
    <div>@ViewBag.Error</div>
}
```

2. **Implement form validation on the views using Razor syntax and client-side validation.**

```
@model RegisterViewModel

<h2>Register</h2>

<form asp-action="Register" asp-controller="Account">
    <div>
        <label asp-for="Username"></label>
        <input asp-for="Username" />
        <span asp-validation-for="Username"></span>
    </div>
    <div>
        <label asp-for="Password"></label>
        <input asp-for="Password" type="password" />
        <span asp-validation-for="Password"></span>
    </div>
    <div>
        <label asp-for="ConfirmPassword"></label>
        <input asp-for="ConfirmPassword" type="password" />
        <span asp-validation-for="ConfirmPassword"></span>
    </div>
    <button type="submit">Register</button>
</form>
```

Task 4: Implement CRUD Operations

1. **Create: Add a new student to the database.**

```
public IActionResult Add()
{
    return View();

}
```

2. **Read: Retrieve a list of all students and display them on the index page.**

```
public IActionResult Delete(int Id)
{
    var studentToUpdate = _db.Students.FirstOrDefault(student => student.Id == Id);

    if(studentToUpdate == null){
        return NotFound();
    }

    return View(studentToUpdate);

}
```

3. **Update: Edit existing student details.**

```
[HttpPost]
0 references
public IActionResult Delete(Students students)
{
    _db.Students.Remove(students);
    _db.SaveChanges();
    return RedirectToAction("Index");

}
```

4. Delete: Remove a student from the database.

```
0 references
public IActionResult Edit(int Id)
{
    var studentToUpdate = _db.Students.FirstOrDefault(students => students.Id == Id);

    if(studentToUpdate == null){
        return NotFound();
    }

    return View(studentToUpdate);

}
```

Task 5: API Integration

1. **Create a separate controller for API calls to fetch student data.**
2. **2. Implement API endpoints for CRUD operations to interact with student data using HTTP methods (GET, POST, PUT, and DELETE).**

```
namespace Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class ApiProductsController : ControllerBase
    {
        11 references
        private readonly AppDbContext _context;

        0 references
        public ApiProductsController(AppDbContext context)
        {
            _context = context;
        }
```

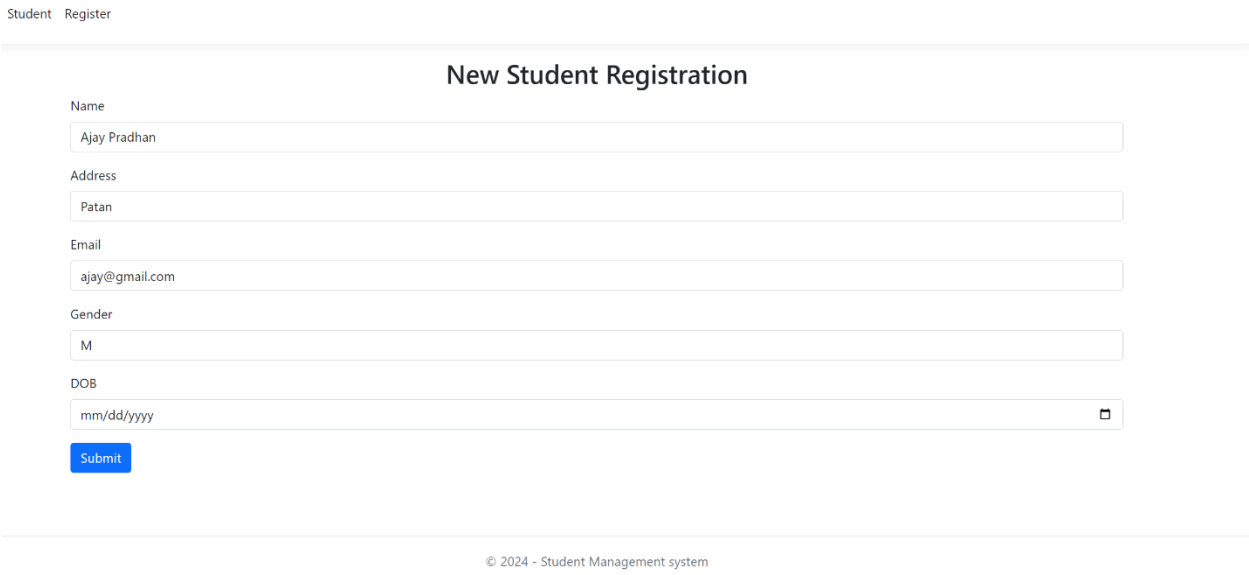
```
[HttpPost]
0 references
public async Task<ActionResult<Product>> PostProduct(Product product)
{
    _context.Products.Add(product);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetProduct), new { id = product.Id }, product);
}
```

```
[HttpDelete("{id}")]
0 references
public async Task<IActionResult> DeleteProduct(int id)
{
    var product = await _context.Products.FindAsync(id);
    if (product == null)
    {
        return NotFound();
    }

    _context.Products.Remove(product);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

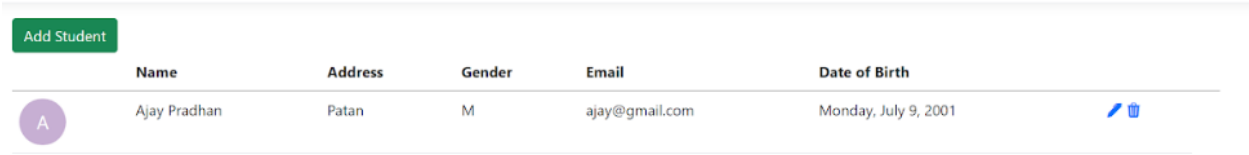
Task 6: Testing

1. **Test the application by adding, editing, and deleting student records from the views.**

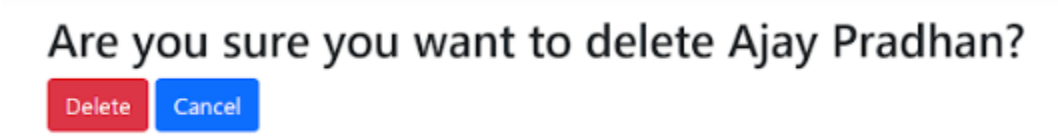
Adding Students



Editing Students



Deleting Students



Task 7: Authentication and Authorization

1. **Implement authentication and authorization mechanisms to control access to the student management system.**

Code:

```
app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();
```

Register:

Program.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<StudentsDB>(options => options.UseSqlite("Data Source = Students.db"));
builder.Services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<StudentsDB>()
    .AddDefaultTokenProviders();

var app = builder.Build();


if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Author}/{action=Index}/{id?}");

app.Run();
```