

TRIBHUVAN U NIVERSITY

VEDAS COLLEGE

**JAWALAKHEL, LALITPUR**

**NEPAL**

**LAB REPORT**

***Subject: Net Centric Computing (CSC 367)***

Submitted By: Submitted To:

Name: **Sadik Ghimire** Department of BSc. CSIT

Roll No: **35** (Bishnu Rawal; Lecturer/Supervisor)

Faculty: **BSc.CSIT**

Year/Sem: III/VI …………………………………………

Table of Content

Net Centric Computing (CSC 367)

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab No.** | **Title** | **Submission Date** | **Signature** |
| 1. | Assignment 1: Starter |  |  |
| 2. | Assignment 2: OOP Concepts |  |  |
| 3. | Assignment 3: File Handling and LINQ |  |  |
| 4. | Assignment 4: ASP .NET Core MVC |  |  |

**Assignment 1: Starter**

**Assignment Description:**

In this assignment, we created a console application named "Greeter" under the "Assignment 1" folder. The application collects user input for their full name and date of birth, then processes and displays this information in a user-friendly manner, including the user's age calculated from their date of birth.

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

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

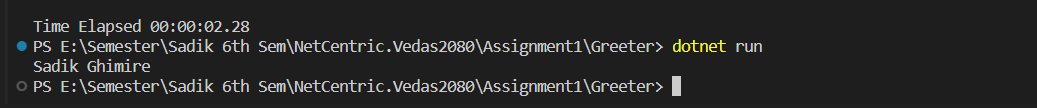
Print value of fullName to console.

**Implementation**

string fullName = “Sadik Ghimire”;

Console.WriteLine(fullName);

**Output Snapshots**



1. Define another variable cFullName and initilize it with fullName in uppercase letters.

Print value of cFullName to console in format: "Hello, Sadik Ghimire Ji!"

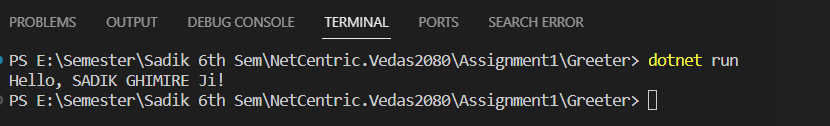
**Implementation**

string fullName = “Sadik Ghimire”;

string cFullName = fullName.ToUpper();

Console.WriteLine($“Hello, { cFullName } Ji!”);

**Output Snapshots**

**

1. Instead of initializing fullName, get it from user.

**Implementation**

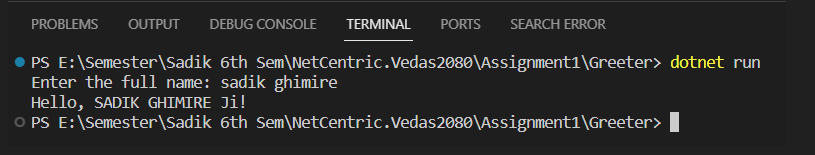
Console.Write("Enter the full name: ");

string fullName= Console.ReadLine();

string cFullName=fullName.ToUpper();

Console.WriteLine($“Hello, { cFullName } Ji!”);

**Output Snapshots**



1. Now also ask user to enter his/her "Date of Birth" and display user friendly date to console. At this point your output should look like assuming user enters dob as "2010/12/12"

**Implementation**

Console.Write("Enter the full name: ");

string fullName= Console.ReadLine();

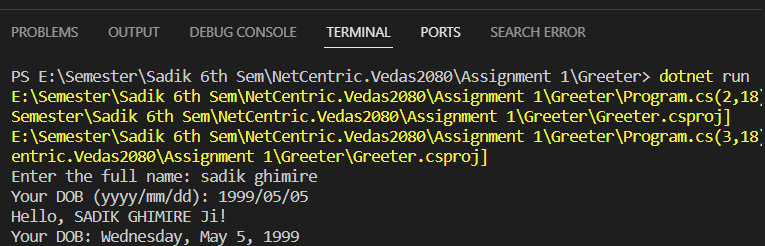
string cFullName=fullName.ToUpper();

Console.Write("Your DOB (yyyy/mm/dd): ");

DateTime.TryParse(Console.ReadLine(), out DateTime dob);

Console.WriteLine($"Hello, {cFullName} Ji! \nYour DOB: {dob.ToLongDateString()}");

**Output Snapshots**



1. Your last task is to calculate his/her age as accurate as possible and display it to console.

**Implementation**

Console.Write("Enter the full name: ");

string fullName= Console.ReadLine();

string cFullName=fullName.ToUpper();

Console.Write("Your DOB (yyyy/mm/dd): ");

DateTime.TryParse(Console.ReadLine(), out DateTime dob);

var ageOutput=CalculateAge(dob);

Console.WriteLine($"Hello, {cFullName} Ji! \nYour DOB: {dob.ToLongDateString()} \n{ageOutput}");

string CalculateAge(DateTime dob){

    // Calculate the age

    TimeSpan difference = DateTime.Now - dob;

    var days = difference.TotalDays;

    var years = (int)(days / 365);

    var remainingDays = days % 365;

    var months =(int)(remainingDays / 30);

    var remainingDaysInMonth = (int)(remainingDays % 30);

    var weeks=(int)(remainingDaysInMonth/7);

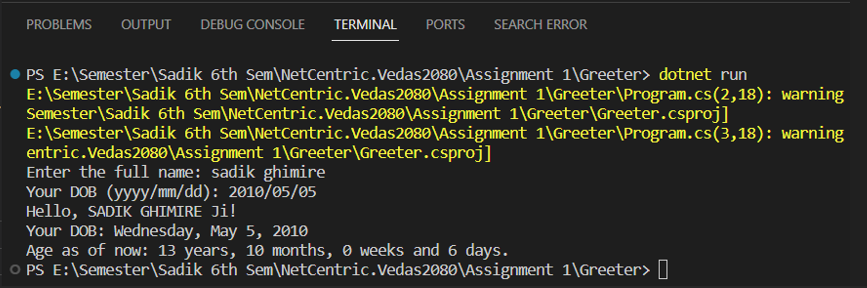
    var remainingDaysInWeek=(int)(remainingDaysInMonth%7);

    var output =$"Age as of now: {years} years, {months} months, {weeks} weeks and {remainingDaysInMonth} days.";

    return output;

}

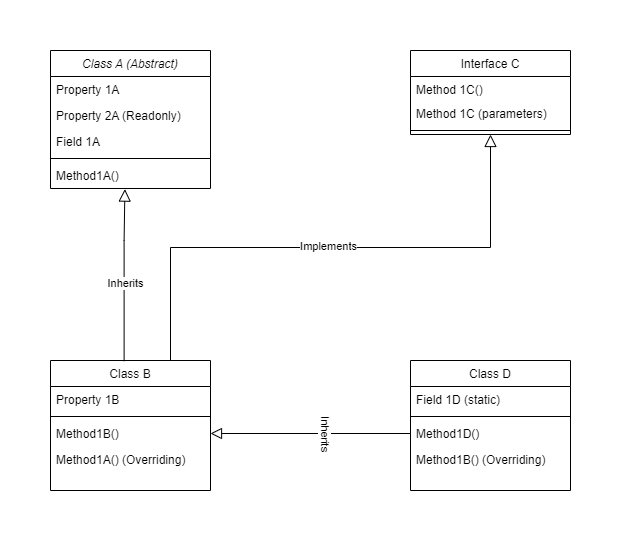
**Output Snapshots**



**Assignment 2: OOP Concepts**

**Assignment Description:**

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

****

**Implementation**

Class A(Library Entity)

public abstract class LibraryEntity

{

    public string? Name { get; set; }

    public abstract string EntityType { get; } // Read-only property

    public string? Description;

    public abstract void DisplayInfo();

}

Class B(Book)

public class Book : LibraryEntity

{

    public string Author { get; set; }

    public int Pages { get; set; }

    public override string EntityType => "Book";

    public override void DisplayInfo()

    {

        Console.WriteLine($"{EntityType}: {Name}");

        Console.WriteLine($"Author: {Author}");

        Console.WriteLine($"Pages: {Pages}");

        Console.WriteLine($"Description: {Description}");

    }

}

Interface C(Librarian)

public interface Librarian

{

void IssueBook(Book book);

void ReturnBook(Book book);

}

Class D(Librarian Staff)

public class LibrarianStaff : LibraryEntity, Librarian

{

public void IssueBook(Book book)

{

Console.WriteLine($"Issuing book {book.Name}");

}

public void ReturnBook(Book book)

{

Console.WriteLine($"Returning book {book.Name}");

}

public override string EntityType => "Librarian Staff";

public override void DisplayInfo()

{

Console.WriteLine($"{EntityType}: {Name}");

Console.WriteLine($"Description: {Description}");

}

}

Program.cs

Book book = new Book

{

Name = "China Harayeko MAnche",

Author = "Hari Bansha Acharya",

Pages = 280,

Description = "The struggles and challenges of a man named Drishya during Nepal's Maoist insurgency."

};

LibrarianStaff librarian = new LibrarianStaff

{

Name = "Sadik Ghimire",

Description = "Senior Librarian"

};

book.DisplayInfo();

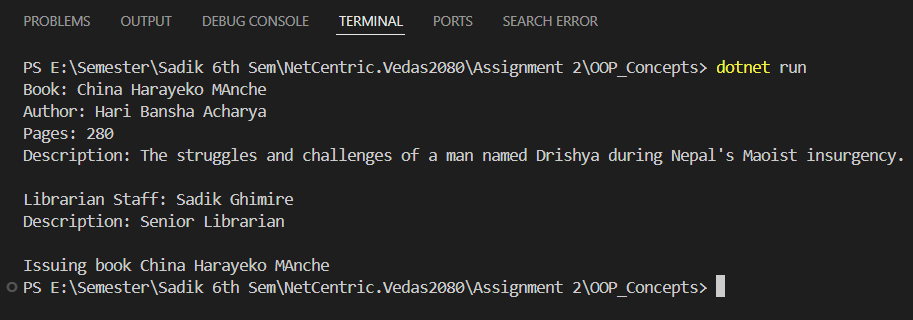
Console.WriteLine();

librarian.DisplayInfo();

Console.WriteLine();

librarian.IssueBook(book);

**Output Snapshots**



**Assignment 3: File Handling and LINQ**

**Assignment Description:**

In this assignment, we created a program to read inflation data from a CSV file, store it in a list of Inflation objects, and perform various analyses on this data. The program consists of two main classes: Inflation and InflationAnalysis.

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

**Implementation**

public class Inflation

{

    public string? RegionalMember { get; set; }

    public int? Year { get; set; }

    public double? InflationRate { get; set; }

    public string? UnitOfMeasurement { get; set; }

    public string? Subregion { get; set; }

    public string? CountryCode { get; set; }

}

1. Create another class InflationAnalysis with methods as needed to
2. Read csv text file and populate List<Inflation> collection with the data read.

**Implementation**

InflationAnalysis class

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

public class InflationAnalysis

{

public List<Inflation> Inflations { get; set; } = new List<Inflation>();

// Method to read CSV file and populate the list

public void ReadCsv(string filePath)

{

var lines = File.ReadAllLines(filePath);

foreach (var line in lines.Skip(1)) // Skip header

{

var values = ParseCsvLine(line);

// Ensure we have the correct number of columns

if (values.Length < 6)

{

continue;

}

try

{

var inflation = new Inflation

{

RegionalMember = values[0].Trim(),

Year = int.Parse(values[1].Trim()),

InflationRate = string.IsNullOrWhiteSpace(values[2]) ? 0 : double.Parse(values[2].Trim(), CultureInfo.InvariantCulture),

UnitOfMeasurement = values[3].Trim(),

Subregion = values[4].Trim(),

CountryCode = values[5].Trim()

};

Inflations.Add(inflation);

}

catch (FormatException ex)

{

Console.WriteLine($"Error parsing line: {line}. Exception: {ex.Message}");

}

}

}

// Helper method to parse CSV line considering quoted fields

private string[] ParseCsvLine(string line)

{

var values = new List<string>();

bool inQuotes = false;

string value = "";

foreach (var c in line)

{

if (c == '"' && !inQuotes)

{

inQuotes = true;

continue;

}

if (c == '"' && inQuotes)

{

inQuotes = false;

continue;

}

if (c == ',' && !inQuotes)

{

values.Add(value);

value = "";

continue;

}

value += c;

}

values.Add(value);

return values.ToArray();

}

}

Main Program

using System;

class Program

{

  static void Main(string[] args)

  {

    string filePath = @"E:\Semester\Sadik 6th Sem\NetCentric.Vedas2080\Assignment 3\Inflation.csv";

    InflationAnalysis analysis = new InflationAnalysis();

    // Read CSV file

    analysis.ReadCsv(filePath);

    // Print the data to verify

    foreach (var inflation in analysis.Inflations)

    {

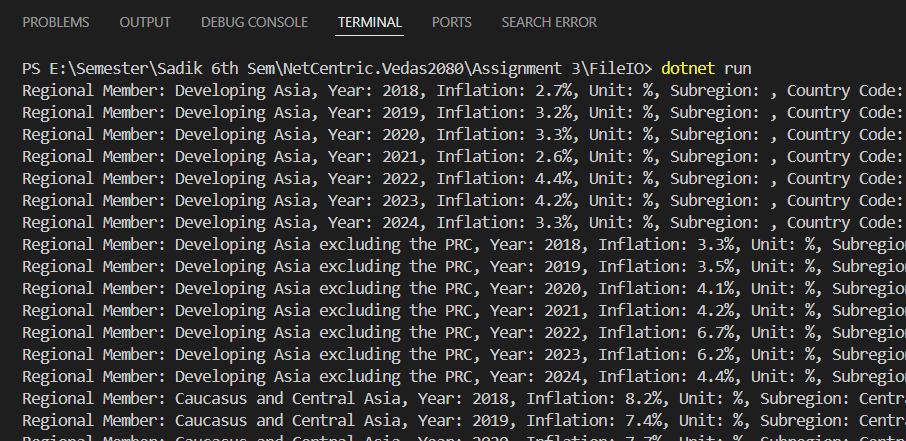
      Console.WriteLine($"Regional Member: {inflation.RegionalMember}, Year: {inflation.Year}, Inflation: {inflation.InflationRate}%, Unit: {inflation.UnitOfMeasurement}, Subregion: {inflation.Subregion}, Country Code: {inflation.CountryCode}");

    }

  }

}

**Output Snapshots**

****

1. To answer following queries related to inflation
   1. Find inflation rates for countries for the year 2021.

**Implementation**

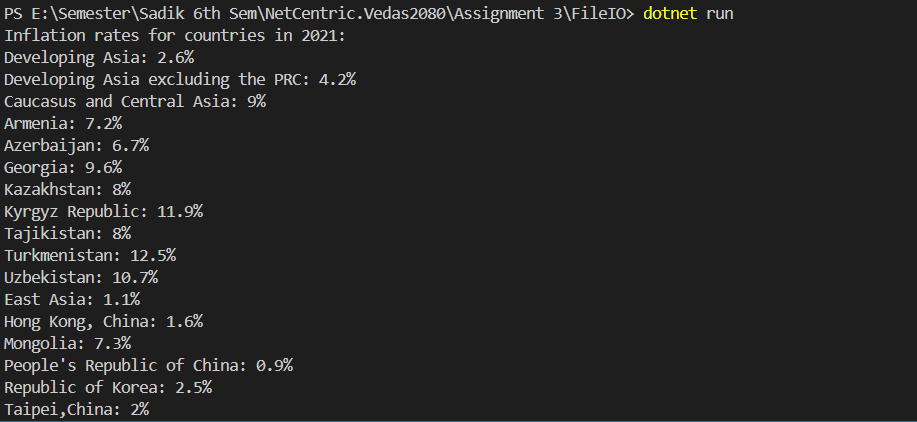
var inflation2021 = analysis.Inflations.Where(i => i.Year == 2021).ToList();

Console.WriteLine("Inflation rates for countries in 2021:");

inflation2021.ForEach(i => Console.WriteLine($"{i.RegionalMember}:

{i.InflationRate}%"));

**Output Snapshots**



* 1. A year when Nepal has highest inflation.

**Implementation**

var nepalInflation = analysis.Inflations

            .Where(i => i.RegionalMember == "Nepal")

            .OrderByDescending(i => i.InflationRate)

            .FirstOrDefault();

        if (nepalInflation != null)

        {

            Console.WriteLine($"Nepal had the highest inflation rate of {nepalInflation.InflationRate}% in the year {nepalInflation.Year}.");

        }

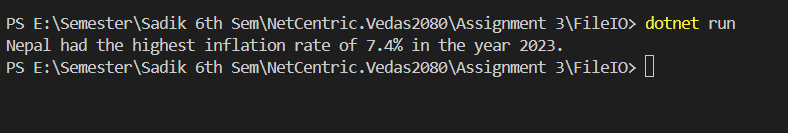
        else

        {

            Console.WriteLine("No data available for Nepal.");

        }

**Output Snapshots**



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

**Implementation**

var top10Countries = analysis.Inflations

.GroupBy(i => i.RegionalMember)

.Select(g => new

{

Country = g.Key,

MaxInflation = g.Max(i => i.InflationRate)

})

.OrderByDescending(g => g.MaxInflation)

.Take(10)

.ToList();

// Print the top 10 countries with highest inflation rates

Console.WriteLine("Top 10 regions (countries) with highest inflation rates:");

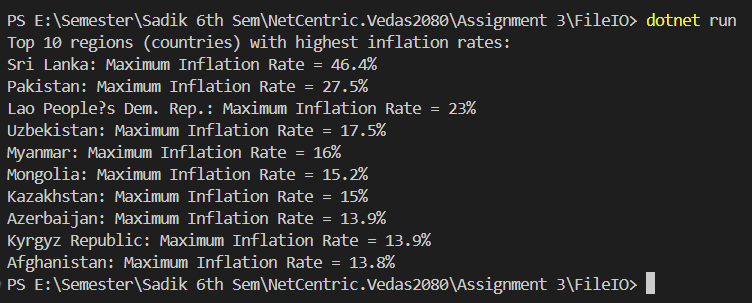
foreach (var item in top10Countries)

{

Console.WriteLine($"{item.Country}: Maximum Inflation Rate = {item.MaxInflation}%");

}

**Output Snapshots**



* 1. List top 3 south Asian countries with lowest inflation rate for year 2020

**Implementation**

var top3Countries = analysis.Inflations

            .Where(i => i.Subregion == "South Asia" && i.Year == 2020)

            .OrderBy(i => i.InflationRate)

            .Take(3)

            .ToList();

        // Print the top 3 countries with lowest inflation rates

        Console.WriteLine("Top 3 South Asian countries with lowest inflation rates for 2020:");

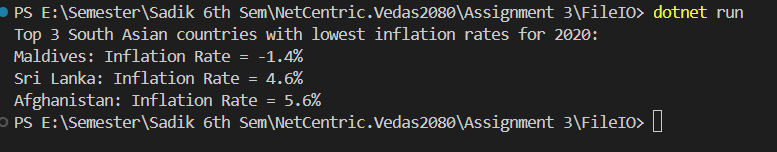
        foreach (var item in top3Countries)

        {

            Console.WriteLine($"{item.RegionalMember}: Inflation Rate = {item.InflationRate}%");

        }

**Output Snapshots**



**Assignment 4: ASP .NET Core MVC**

**Assignment Description:**

This assignment integrate the Student Management System project in ASP .NET Core MVC involves setting up, modeling, and implementing CRUD operations, integrating APIs, testing, and implementing secure authentication and authorization features. This comprehensive system aims to efficiently manage student data while ensuring security and usability.

Task 1: Setting Up the Project

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

dotnet new mvc -n StudentManagementSystem

1. Install necessary NuGet packages for SQLite and API calls

dotnet add package Microsoft.EntityFrameworkCore.Sqlite

dotnet add package System.Net.Http.Json

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.EntityFrameworkCore

dotnet tool install --global dotnet-ef

1. Set up SQLite database context for student management.

using Microsoft.EntityFrameworkCore;

using StudentManagementSystem.Models;

namespace StudentManagementSystem.Data

{

public class StudentDbContext : DbContext

{

public StudentDbContext(DbContextOptions<StudentDbContext> options) : base(options) { }

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

}

}

Task 2: Creating Models

using System.ComponentModel.DataAnnotations;

namespace StudentManagementSystem.Models

{

public class Student

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

[EmailAddress]

public string Email { get; set; }

[Range(18, 100)]

public int Age { get; set; }

}

}

Task 3: Creating Views

@model IEnumerable<StudentManagementSystem.Models.Student>

  @{

      ViewData["Title"] = "Students";

  }

  <h1>Students</h1>

  <p>

      <a asp-action="Create">Create New</a>

  </p>

  <table class="table">

      <thead>

          <tr>

              <th>

                  @Html.DisplayNameFor(model => model.Name)

              </th>

              <th>

                  @Html.DisplayNameFor(model => model.Email)

              </th>

              <th>

                  @Html.DisplayNameFor(model => model.Age)

              </th>

              <th>

                 Action

              </th>

          </tr>

      </thead>

      <tbody>

          @foreach (var item in Model)

          {

              <tr>

                  <td>

                      @Html.DisplayFor(modelItem => item.Name)

                  </td>

                  <td>

                      @Html.DisplayFor(modelItem => item.Email)

                  </td>

                  <td>

                      @Html.DisplayFor(modelItem => item.Age)

                  </td>

                  <td>

                      <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

                      <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>

                  </td>

              </tr>

          }

      </tbody>

  </table>

Task 4: Controller

using Microsoft.AspNetCore.Mvc;

using StudentManagementSystem.Data;

using StudentManagementSystem.Models;

using System.Linq;

namespace StudentManagementSystem.Controllers

{

    public class StudentController : Controller

    {

        private readonly StudentDbContext \_context;

        public StudentController(StudentDbContext context)

        {

            \_context = context;

        }

        public IActionResult Index()

        {

            var students = \_context.Students.ToList();

            return View(students);

        }

        public IActionResult Create()

        {

            return View();

        }

        [HttpPost]

        public IActionResult Create(Student student)

        {

            if (ModelState.IsValid)

            {

                \_context.Students.Add(student);

                \_context.SaveChanges();

                return RedirectToAction(nameof(Index));

            }

            return View(student);

        }

        public IActionResult Edit(int id)

        {

            var student = \_context.Students.Find(id);

            if (student == null)

            {

                return NotFound();

            }

            return View(student);

        }

        [HttpPost]

        public IActionResult Edit(Student student)

        {

            if (ModelState.IsValid)

            {

                \_context.Students.Update(student);

                \_context.SaveChanges();

                return RedirectToAction(nameof(Index));

            }

            return View(student);

        }

        public IActionResult Delete(int id)

        {

            var student = \_context.Students.Find(id);

            if (student == null)

            {

                return NotFound();

            }

            return View(student);

        }

        [HttpPost, ActionName("Delete")]

        public IActionResult DeleteConfirmed(int id)

        {

            var student = \_context.Students.Find(id);

            if (student == null)

            {

                return NotFound();

            }

            \_context.Students.Remove(student);

            \_context.SaveChanges();

            return RedirectToAction(nameof(Index));

        }

    }

}

Task 5: Implement CRUD Operations

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

@model StudentManagementSystem.Models.Student

@{

ViewData["Title"] = "Create Student";

}

<h1>Create Student</h1>

<h4>Student</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div class="form-group">

<label asp-for="Name" class="control-label"></label>

<input asp-for="Name" class="form-control" />

<span asp-validation-for="Name" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Email" class="control-label"></label>

<input asp-for="Email" class="form-control" />

<span asp-validation-for="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Age" class="control-label"></label>

<input asp-for="Age" class="form-control" />

<span asp-validation-for="Age" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

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

@model IEnumerable<StudentManagementSystem.Models.Student>

@{

ViewData["Title"] = "Students";

}

<h1>Students</h1>

<p>

<a asp-action="Create">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th>

Action

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

<a asp-action="Delete" asp-route-id="@item.Id">Delete</a>

</td>

</tr>

}

</tbody>

</table>

1. Update: Edit existing student details.

@model StudentManagementSystem.Models.Student

@{

ViewData["Title"] = "Edit Student";

}

<h1>Edit Student</h1>

<h4>Student</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div class="form-group">

<label asp-for="Name" class="control-label"></label>

<input asp-for="Name" class="form-control" />

<span asp-validation-for="Name" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Email" class="control-label"></label>

<input asp-for="Email" class="form-control" />

<span asp-validation-for="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Age" class="control-label"></label>

<input asp-for="Age" class="form-control" />

<span asp-validation-for="Age" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Delete: Remove a student from the database.

@model StudentManagementSystem.Models.Student

@{

ViewData["Title"] = "Delete Student";

}

<h1>Delete Student</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Student</h4>

<hr />

<dl class="row">

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Name)

</dd>

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Email)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Email)

</dd>

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Age)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Age)

</dd>

</dl>

<form asp-action="Delete">

<input type="hidden" asp-for="Id" />

<input type="submit" value="Delete" class="btn btn-danger" /> |

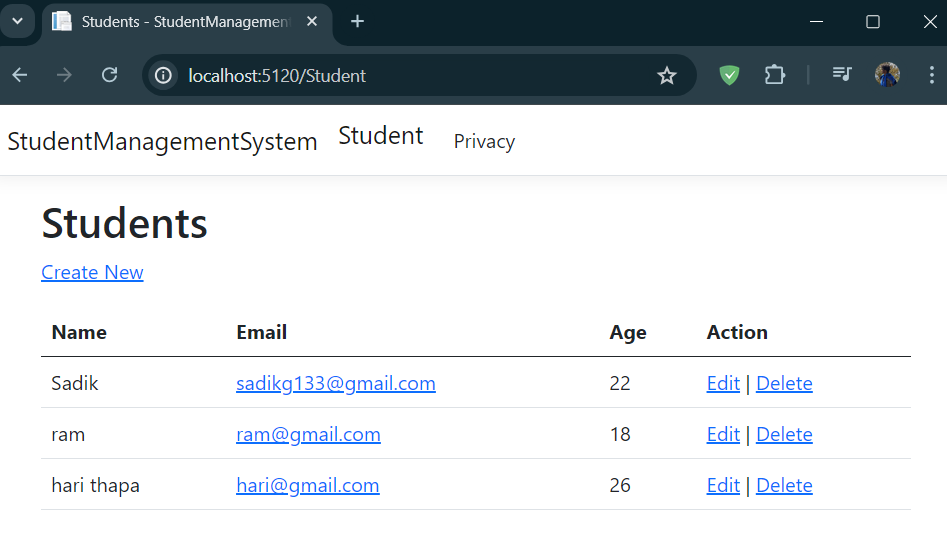
<a asp-action="Index">Back to List</a>

</form>

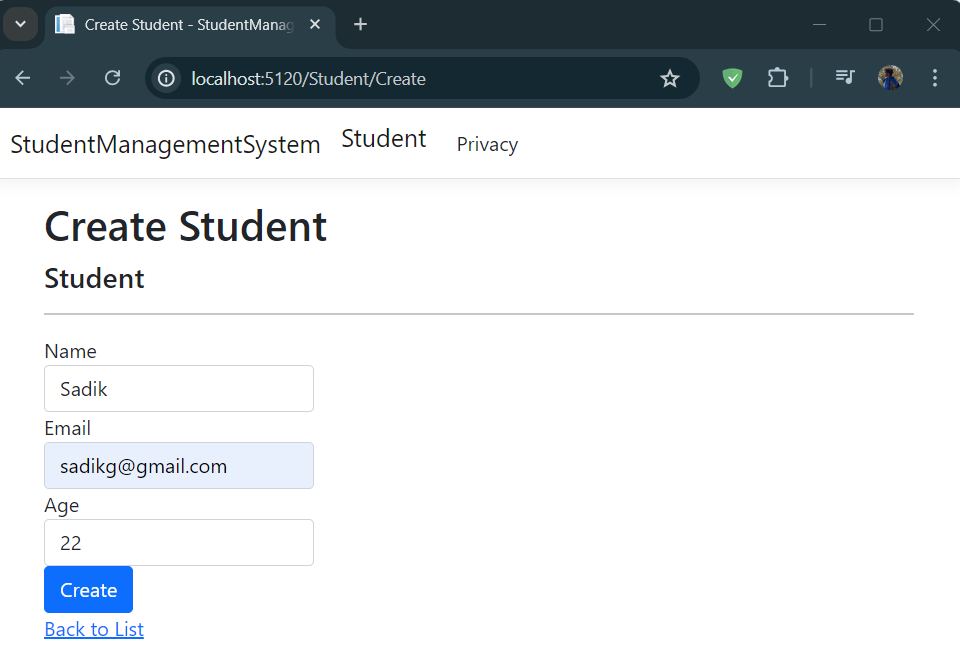
</div>

**Output Snapshots**

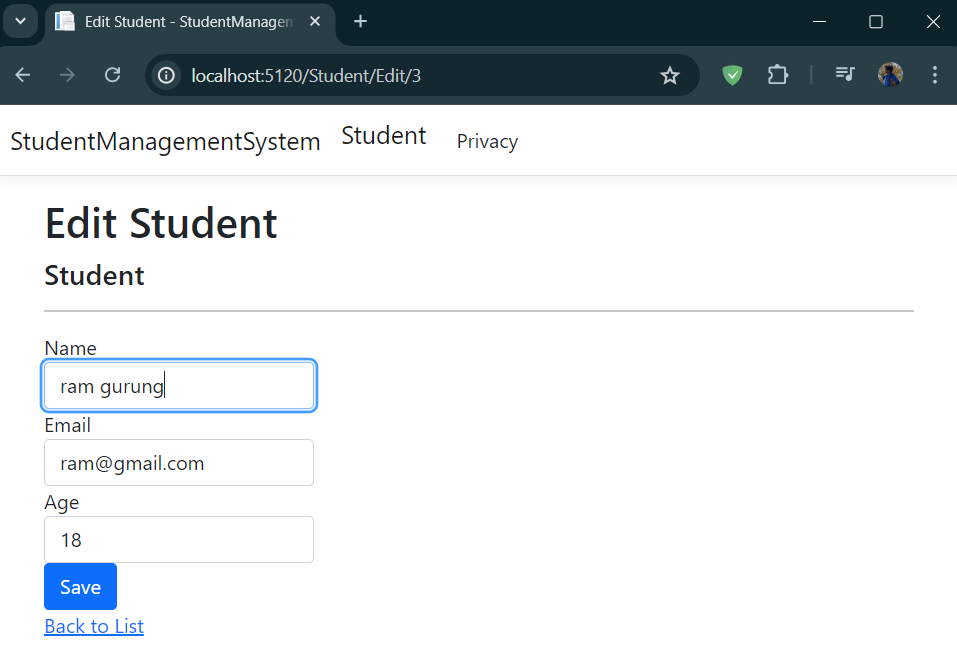
Index page:



Add new student:



Edit Student



Delete Student

