ASSIGNMENT 1(Starter)

Introduction:

This lab report focuses on creating a C# console application named Greeter that demonstrates the use of variables, string manipulation, and user input. Initially, a name is printed using a predefined variable, followed by converting it to uppercase with a personalized greeting. The program then collects the user's full name and date of birth, displays the formatted date, and calculates the precise age using date-time operations. This step-by-step exercise helps reinforce basic concepts in C# programming.

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

Description:

In the code below, a string variable named fullName is initialized with value “Anish Shrestha”. Afterward, the variable is displayed using Console.WriteLine().

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Greeter
{

internal class Program
   {

static void Main(string[] args)
       {
           string fullname = "Anish Shrestha";
           Console.WriteLine("Full name: " + fullname); Console.ReadLine();
       }
```

```
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Greeter
{

    internal class Program
    {

        static void Main(string[] args)
        {
            string fullname = "Anish Shrestha";
            string cFullname = fullname.ToUpper();
            Console.WriteLine($"Hello, {cFullname} Ji!");
            Console.ReadLine();
        }
    }
}
```

Description:

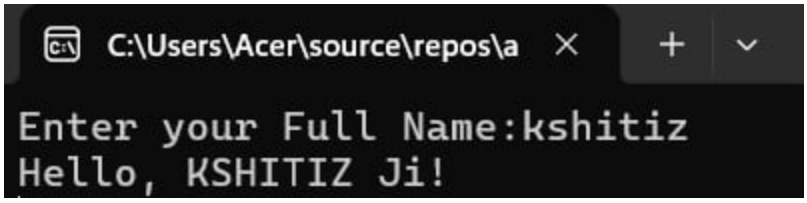
A variable named cFullName is defined which is initialized with a fullName variable in Uppercase.

Print value of fullName to console.

```
namespace Greeter
{
    internal class Program
    {

        static void Main(string[] args)
        {
            Console.Write("Enter your Full Name:");
            string fullname = Console.ReadLine();
            string cFullname = fullname.ToUpper();
            Console.WriteLine($"Hello, {cFullname} Ji!");
            Console.ReadLine();
        }
    }
}
```

Description:

Instead of initializing the value of fullName, reading it from the console using Console.ReadLine().

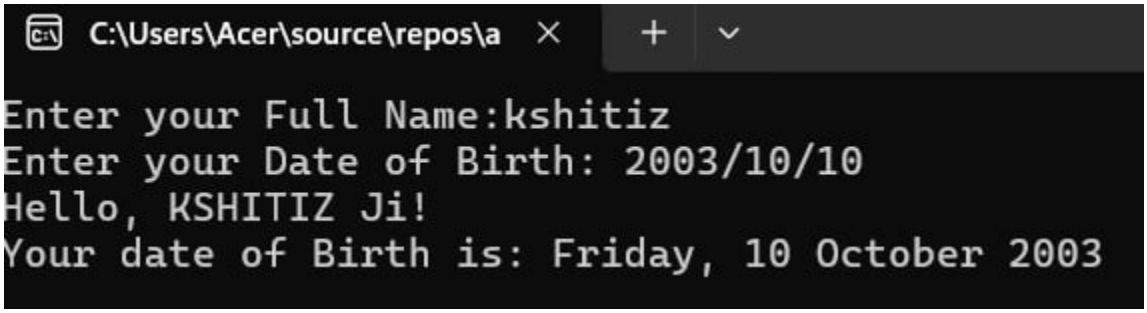


Define another variable cFullName and initilize it with fullName in uppercase letters & ask for DOB and format it nicely

```
namespace Greeter
{
    internal class Program
    {
        static void Main(string[] args)
        {

            Console.Write("Enter your Full Name:");
            string fullname = Console.ReadLine();
            string cFullname = fullname.ToUpper();
            Console.Write("Enter your Date of Birth: ");
            string dobinput = Console.ReadLine();
            DateTime dob = DateTime.Parse(dobinput);
            string formatteddob = dob.ToString("dddd, dd MMMM yyyy");
            Console.WriteLine($"Hello, {cFullname} Ji!");
            Console.WriteLine($"Your date of Birth is: {formatteddob}");
            Console.ReadLine();
        }
    }

}
```

Description:

Taking input date of birth along with full name from user and displaying in console. The date of birth is displayed by parsing using DateTime.parse method.



Calculate his/her age as accurate as possible and dispaly it to console.

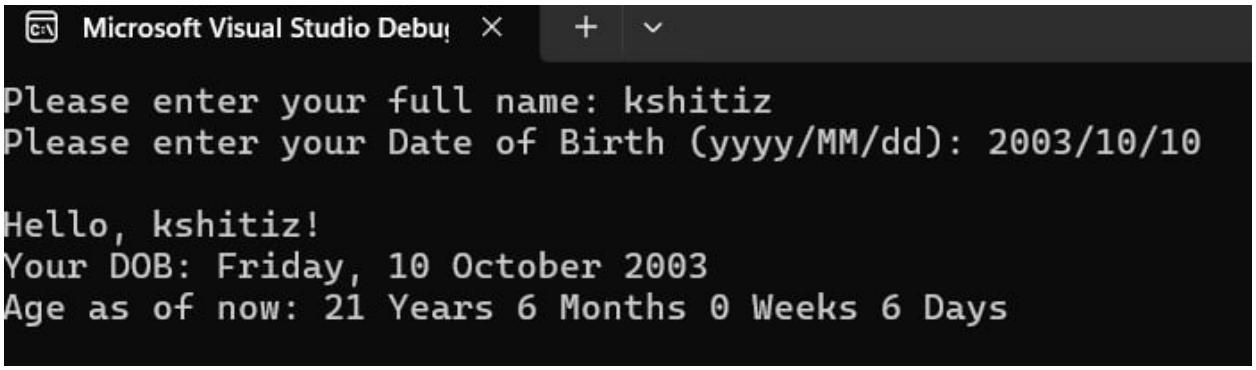
```
using System;

namespace Greeter
{
   class Program
   {
      static void Main(string[] args)
      {

         Console.Write("Please enter your full name: ");
         string fullName = Console.ReadLine();

         Console.Write("Please enter your Date of Birth (yyyy/MM/dd): ");
         string dobInput = Console.ReadLine();

         DateTime dob;
         if (!DateTime.TryParse(dobInput, out dob))
         {
            Console.WriteLine("Invalid Date Format! Please enter the date in yyyy/MM/dd
format.");
            return;
```

```
        }

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

        DateTime now = DateTime.Now;

        int years = now.Year - dob.Year;
        int months = now.Month - dob.Month;
        int days = now.Day - dob.Day;

        if (days < 0)
        {
            months--;
            int previousMonth = now.Month == 1 ? 12 : now.Month - 1;
            int yearForPreviousMonth = now.Month == 1 ? now.Year - 1 : now.Year;
            days += DateTime.DaysInMonth(yearForPreviousMonth, previousMonth);
        }
        if (months < 0)
        {
            years--;
            months += 12;
        }

        int weeks = days / 7;
        int remainingDays = days % 7;

        Console.WriteLine("Age as of now: {0} Years {1} Months {2} Weeks {3} Days",
years, months, weeks, remainingDays);
      }
    }
}
```

Description:

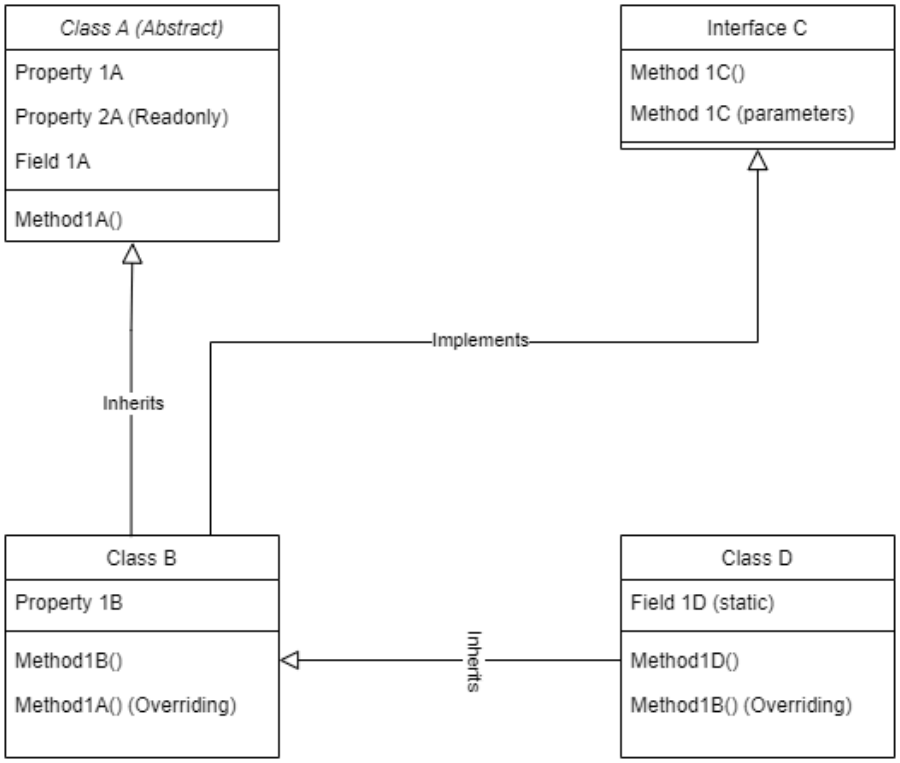
Calculating the age of the user by subtracting the date of birth from the current date. Firstly, the date of birth of the user is read from the

terminal. The age as of now is calculated by performing arithmetic calculations using division and modulo.



ASSIGNMENT 2(OOP Concepts)

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



Introduction:

This assignment explores the core concepts of Object-Oriented Programming (OOP) through a real-world scenario modeled using abstract classes, inheritance, interfaces, properties, methods, and overriding. The provided class diagram illustrates a structure where an abstract base class defines common members, which are then inherited and specialized by derived classes. An interface is also introduced to ensure certain

methods are implemented across classes. This structure helps in understanding the pillars of OOP such as abstraction, encapsulation, inheritance, polymorphism, and interface implementation.

Task 1

Description:

The Vehicle abstract class is created to define common properties like Brand, Model, and a readonly MaxSpeed. It includes an abstract method StartEngine() to be implemented by derived classes.

```
using System;
1 reference
abstract class Vehicle
{
    1 reference
    public string Brand { get; set; }
    1 reference
    public string Model { get; set; }
    public readonly int MaxSpeed = 200;
    1 reference
    public abstract void StartEngine();
}
```

Task 2

Description:

The IFuelEfficiency interface is created to define methods for calculating and displaying fuel efficiency, ensuring any implementing class provides specific logic for both.

```
1 reference
interface IFuelEfficiency
{
    1 reference
    void CalculateFuelEfficiency();
    2 references
    void DisplayEfficiency(string fuelType);
}
```

Task 3

Description:

The Car class inherits from the Vehicle abstract class and implements the IFuelEfficiency interface. It adds a specific property NumberOfDoors and provides its own implementation of StartEngine, along with methods to calculate and display fuel efficiency.

```
class Car : Vehicle, IFuelEfficiency
{
    0 references
    public int NumberOfDoors { get; set; }

    1 reference
    public override void StartEngine()
    {
        Console.WriteLine("Car engine started with a key!");
    }

    1 reference
    public void CalculateFuelEfficiency()
    {
        Console.WriteLine("Calculating fuel efficiency for the car...");
    }

    2 references
    public void DisplayEfficiency(string fuelType)
    {
        Console.WriteLine($"Car fuel type: {fuelType}");
    }
}
```

Task 4

Description:

The ElectricCar class inherits from Car and introduces a static field BatteryCapacity. It adds a new method ChargeBattery() and uses the new keyword to hide the base class StartEngine() method, providing a specialized implementation for electric vehicles.

```
class ElectricCar : Car
{
    public static int BatteryCapacity = 100;  // Static field

    1 reference
    public void ChargeBattery()
    {
        Console.WriteLine("Charging electric car...");
    }

    1 reference
    public new void StartEngine()
    {
        Console.WriteLine("Electric car starts silently!");
    }
}
```

Task 5

Description:

The Program class contains the Main method, which creates an instance of ElectricCar. It sets the Brand and Model, calls the overridden StartEngine(), charges the battery, and displays the fuel efficiency, demonstrating inheritance, method hiding, and interface implementation in action.

```
class Program
{
    0 references
    static void Main()
    {
        ElectricCar myTesla = new ElectricCar();
        myTesla.Brand = "Tesla";
        myTesla.Model = "Model S";

        myTesla.StartEngine();
        myTesla.ChargeBattery();
        myTesla.DisplayEfficiency("Electric");
    }
}
```

RESULT:

```
Electric car starts silently!
Charging electric car...
Car fuel type: Electric

C:\Users\Acer\source\repos\ksihitiz\ksihitiz\b
```

ASSIGNMENT 3(File Handling and LINQ)

Objective:

The objective of this assignment is to show how C# can read CSV data through file management features while utilizing LINQ. Our objective is to design an inflation data model using file handling and LINQ for performing searches which will provide insights about time periods with high or low inflation and regions and yearly data breakdown.

Creating `Inflation` Class

```
0 references
public class Inflation
{
    0 references
    public string RegionalMember { get; set; } = "";
    0 references
    public int Year { get; set; }
    0 references
    public double InflationRate { get; set; }
    0 references
    public string UnitOfMeasurement { get; set; } = "";
    0 references
    public string Subregion { get; set; } = "";
    0 references
    public string CountryCode { get; set; } = "";
}
```

Creating `InflationAnalysis` Class

```
0 references
public List<Inflation> ReadCsv(string path)
{
    var lines = File.ReadAllLines("Inflation.csv").Skip(1);
    var data = new List<Inflation>();

    foreach (var line in lines)
    {
        var columns = line.Split(',');

        if (columns.Length < 6) continue;

        try
        {
            data.Add(new Inflation
            {
                RegionalMember = columns[0].Trim('\"'),
                Year = int.Parse(columns[1].Trim('\"')),
                InflationRate = double.Parse(columns[2].Trim('\"')),
                UnitOfMeasurement = columns[3].Trim('\"'),
                Subregion = columns[4].Trim('\"'),
                CountryCode = columns[5].Trim('\"')
            });
        }
        catch
        {
            continue;
        }
    }

    return data;
}
```

Creating `Program` Class

```
class Program
{
    static void Main(string[] args)
    {
        var analyzer = new InflationAnalysis();
        var data = analyzer.ReadCsv("Inflation.csv");
        analyzer.Query(data);
    }
}
```

```
PS C:\Users\Acer\Downloads\InflationAnalysis\InflationAnalysis> dotnet run

 a) Inflation Rates in 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%
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%
```

```
Fiji: 0.2%
Kiribati: 1%
Marshall Islands: 2.2%
Nauru: 1.2%
Palau: 0.5%
Papua New Guinea: 4.5%
Samoa: -3%
Solomon Islands: -0.2%
Tonga: 1.4%
Tuvalu: 6.7%
Vanuatu: 2.3%

 b) Year Nepal had highest inflation: 2023 (7.4%)

 c) Top 10 countries with highest inflation (all time):
Sri Lanka (2022) - 46.4%
Pakistan (2023) - 27.5%
Sri Lanka (2023) - 24.6%
Lao People's Dem. Rep. (2022) - 23%
Lao People's Dem. Rep. (2023) - 16%
Myanmar (2022) - 16%
Mongolia (2022) - 15.2%
Kazakhstan (2022) - 15%
Pakistan (2024) - 15%

Top 3 South Asian countries with lowest inflation in 2020:
Maldives: -1.4%
Sri Lanka: 4.6%
Afghanistan: 5.6%
PS C:\Users\Acer\Downloads\InflationAnalysis\InflationAnalysis>
```

ASSIGNMENT 4(Movie Management System)

Introduction and Theory:
This lab demonstrates the use of MVC (Model View Controller) dot net core application to create a movie management web portal.

What is MVC?
Model-view-controller (MVC) is an architectural design pattern that organizes an application's logic into distinct layers, each of which carries out a specific set of tasks. Asp.net provide us flexible way to create MVC application that implements separation of concerns between the application logics. Let’s understand each design layer of MVC.

Model:
Model layer is responsible for handling the data logic of the application. It represents the corresponding table in our database. The model layer is updated by the controller whereas it is displayed in view layer.

View:
View layer shows the ui layer of the application. The users interact to our application using the view layer. This layer shows the data present in the model layer and is render based on the logic present in the controller. The view might include headings, tables, lists, forms, etc.

Controller:

Controller layer implements the application logic necessary for the communication in the application. It determines how the view are rendered. It acts as the brain of the application synchronizing the communication. It is responsible for updating the model and rendering the view based on the user action.

Movie Model: Create a file named “Movie.cs” in the “models” folder of the root project. The movie class should have following properties.
Movie (ID, Title, Rating, Budget, Gross, Release Date, Genre, Runtime, Summary)

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;

namespace MovieManagementPortal.Models
{
    14 references
    public class Movie
    {
        [Key]
        [JsonPropertyName("id")]
        7 references
        public int Id { get; set; }

        [Required]
        [StringLength(200, ErrorMessage = "Title cannot exceed 200 characters.")]
        [JsonPropertyName("title")]
        8 references
        public string Title { get; set; } = string.Empty;

        [Range(0, 10, ErrorMessage = "Rating must be between 0 and 10.")]
        [JsonPropertyName("rating")]
        7 references
        public int Rating { get; set; }

        [Range(0, double.MaxValue, ErrorMessage = "Budget must be a positive number.")]
        [JsonPropertyName("budget")]
        11 references
        public float Budget { get; set; }

        [Range(0, double.MaxValue, ErrorMessage = "Gross revenue must be a positive number.")]
        [JsonPropertyName("gross")]
        5 references
        public float Gross { get; set; }

        [DataType(DataType.Date)]
        [JsonPropertyName("release_date")]
    public DateTime ReleaseDate { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "Genre cannot exceed 100 characters.")]
    [JsonPropertyName("genre")]
    7 references
    public string Genre { get; set; } = string.Empty;

    [Range(1, 600, ErrorMessage = "RunTime must be between 1 and 600 minutes.")]
    [JsonPropertyName("runtime")]
    7 references
    public float RunTime { get; set; }

    [Required]
    [StringLength(1000, ErrorMessage = "Summary cannot exceed 1000 characters.")]
    [JsonPropertyName("summary")]
    7 references
    public string Summary { get; set; } = string.Empty;
}
```

Actor model: Create a file named “Actor.cs” inside the “models” folder and add a class Actor with following fields and required data annotations.

Actor (ID, Name, Date of Birth, Birth City, Birth Country, Height (Inches), Biography, Gender, Net Worth)

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json.Serialization;

namespace MovieManagementPortal.Models
{
    14 references
    public class Actor
    {
        [Key]
        [JsonPropertyName("id")]
        8 references
        public int Id { get; set; }

        [Required]
        [MaxLength(100)]
        8 references
        public string Name { get; set; }

        [Required]
        7 references
        public DateTime DateOfBirth { get; set; }

        [Required]
        [MaxLength(50)]
        7 references
        public string Country { get; set; }

        [Required]
        [Range(20, 100)]
        7 references
        public float Height { get; set; }

        [MaxLength(500)]
        7 references
        public string Biography { get; set; }
```

```
        [Required]
        [RegularExpression("^(Male|Female|Other)$", ErrorMessage = "Gender must be Male, Female, or Other.")]
        6 references
        public string Gender { get; set; }

        [Required]
        [Column(TypeName = "decimal(18,2)")]
        [Range(0, double.MaxValue, ErrorMessage = "NetWorth cannot be negative.")]
        9 references
        public float NetWorth { get; set; }


        0 references
        public List<Character> Characters { get; set; } = new();
    }
```

Character Model: Create a file named “Character.cs” inside the “models” folder and add a class Character with following fields and required data annotations.

Character (MovieID, ActorID, Character Name, Pay, Screentime)

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace MovieManagementPortal.Models
{
    2 references
    public class Character
    {
        [Key]
        0 references
        public int Id { get; set; }

        [Required]
        0 references
        public int MovieId { get; set; }
        0 references
        public Movie Movie { get; set; }

        [Required]
        0 references
        public int ActorId { get; set; }
        0 references
        public Actor Actor { get; set; }

        [Required]
        [MaxLength(100)]
        0 references
        public string CharacterName { get; set; }

        [Required]
        [Column(TypeName = "decimal(18,2)")]
        0 references
        public decimal Pay { get; set; }

        [Required]
        [Range(0, 300)]
        0 references
        public float Screentime { get; set; }
    }
```

Code on Program.cs

Add the service for the setup of the ApplicationDbContext using Dependency Injection.

```
// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddRazorPages();

builder.Services.AddDbContext<MovieDbContext>(options => options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
```

```
[HttpGet]
0 references
public IActionResult Create()
{

    return View();
}

[HttpPost]

0 references
public async Task<IActionResult> Create(Movie movie)
{

    if (!ModelState.IsValid)
    {
        return View(movie);
    }

    _context.Movies.Add(movie);

    await _context.SaveChangesAsync();
    return RedirectToAction("Index");

}
```

```
public async Task<IActionResult> Index()
{
    return View(await _movieDbContext.Actors.ToListAsync());
}

[HttpGet]
0 references
public IActionResult Create() {

    return View();
}

[HttpPost]
0 references
public async Task<IActionResult> Create(Actor actor) {

    if (ModelState.IsValid) {

        _movieDbContext.Actors.Add(actor);

        await _movieDbContext.SaveChangesAsync();

        return RedirectToAction("Index");
    }

    return View(actor);
}
```

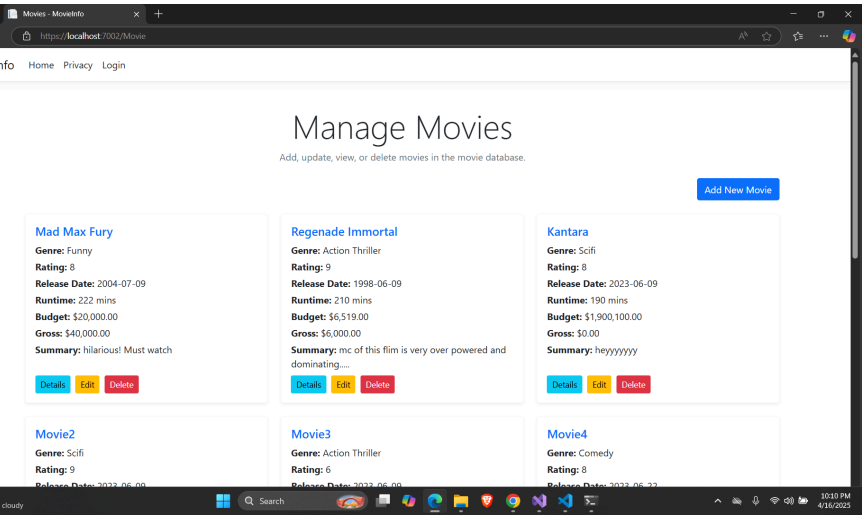
```
public class DashboardController : Controller
{
    private readonly MovieDbContext _context;

    0 references
    public DashboardController(MovieDbContext context)
    {
        _context = context;
    }

    0 references
    public IActionResult Index()
    {
        var topPaidActors = _context.Actors
            .OrderByDescending(a => a.NetWorth)
            .Take(5)
            .ToList();

        var flopMovies = _context.Movies
            .Where(m => m.Gross < m.Budget)
            .OrderBy(m => m.Budget - m.Gross)
            .Take(10)
            .ToList();

        var dashboardViewModel = new DashboardViewModel
        {
            Actors = topPaidActors,
            Movies = flopMovies
        };

        return View(dashboardViewModel);
    }
```

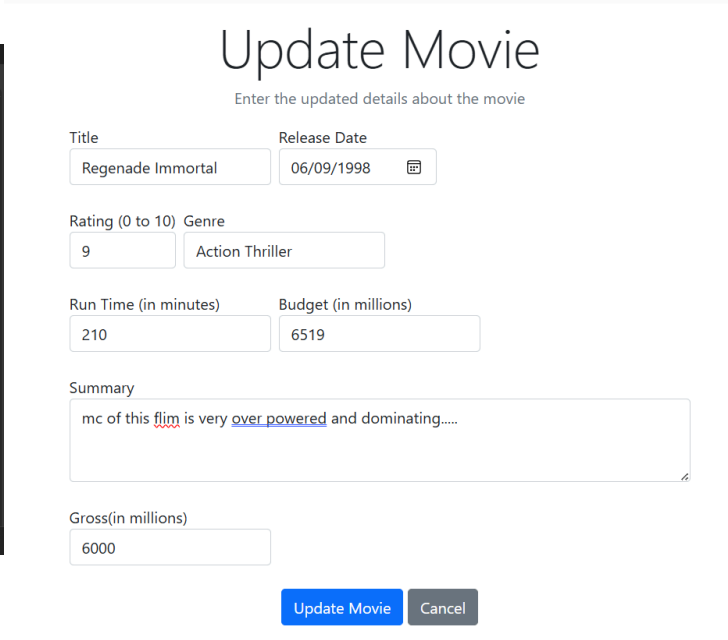
Views/Movie/Create.cshtml

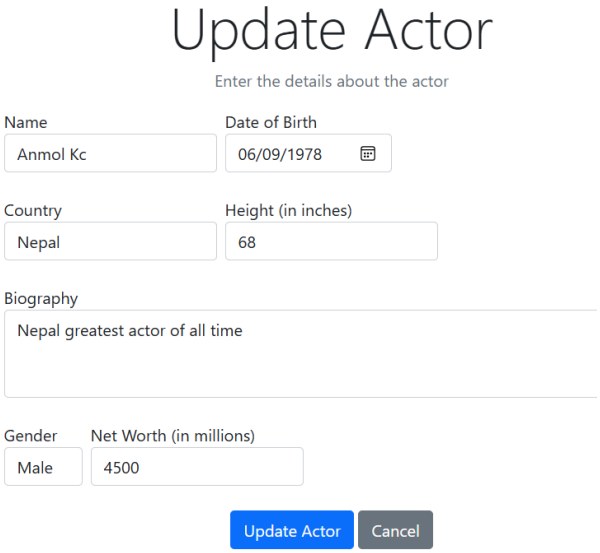
```
Create.cshtml   AccountController.cs   Index.cshtml   Movie.cs   AuthDbContext.cs   Program.cs   MovieController.cs*   Character.cs   MovieDbContext.cs
MovieManagementPortal
@model MovieManagementPortal.Models.Movie
@{
    ViewData["Title"] = "Create Movie";
}

<div class="container mt-4">

    <div class="text-center">
        <h1 class="display-4">Add Movie</h1>
        <p class="text-muted">Enter the details about the movie</p>
    </div>

    <div class="d-flex justify-content-center align-items-center border-2 border-secondary">
        <form class="d-flex gap-0 flex-column m-50" method="post" asp-action="Create" asp-controller="Movie">
            <div class="d-flex gap-2">
                <div class="form-group">
                    <label asp-for="Title">Title</label>
                    <input asp-for="Title" class="form-control" required />
                    <span asp-validation-for="Title" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="ReleaseDate">Release Date</label>
                    <input asp-for="ReleaseDate" type="date" class="form-control" required />
                    <span asp-validation-for="ReleaseDate" class="text-danger"></span>
                </div>
            </div>

            <div class="d-flex gap-2">
                <div class="form-group">
                    <label asp-for="Rating">Rating (0 to 10)</label>
                    <input asp-for="Rating" type="number" min="0" max="10" class="form-control" required />
                    <span asp-validation-for="Rating" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="Genre">Genre</label>
                    <input asp-for="Genre" class="form-control" required />
                    <span asp-validation-for="Genre" class="text-danger"></span>
                </div>
            </div>
```

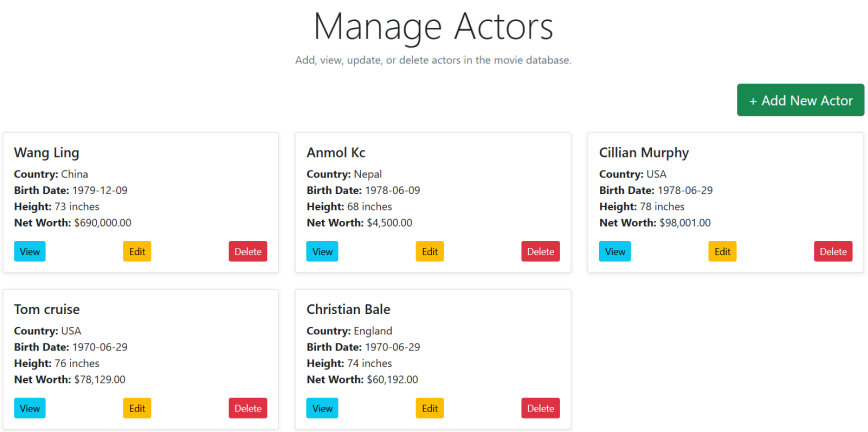
```
            <div class="d-flex gap-2">
                <div class="form-group">
                    <label asp-for="Runtime">Run Time (in minutes)</label>
                    <input asp-for="Runtime" type="number" step="0.1" class="form-control" required />
                    <span asp-validation-for="Runtime" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="Budget">Budget (in millions)</label>
                    <input asp-for="Budget" type="number" step="0.1" class="form-control" required />
                    <span asp-validation-for="Budget" class="text-danger"></span>
                </div>
            </div>

            <div class="d-flex gap-2">
                <div class="form-group m-100">
                    <label asp-for="Summary">Summary</label>
                    <textarea asp-for="Summary" class="form-control" rows="3"></textarea>
                    <span asp-validation-for="Summary" class="text-danger"></span>
                </div>
            </div>

            <div class="text-center">
                <button type="submit" class="btn btn-primary">Create Movie</button>
                <a asp-action="Index" asp-controller="Movie" class="btn btn-secondary">Cancel</a>
            </div>
        </form>
    </div>
</div>
```

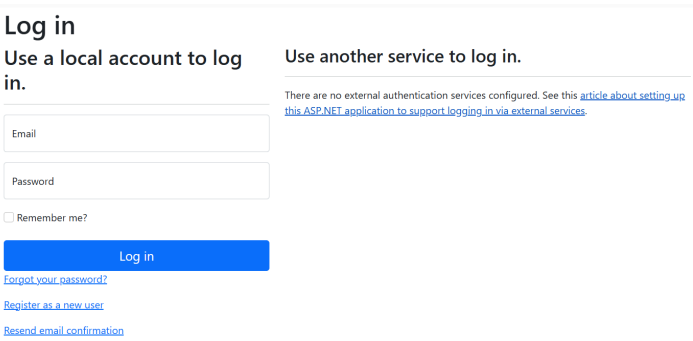
output:

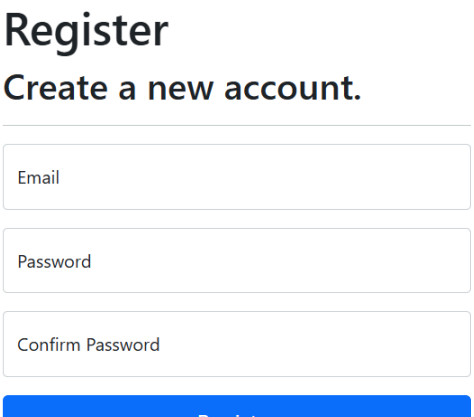

Movies - MovieInfo
https://localhost:7002/Movie
Home Privacy Login
Manage Movies
Add, update, view, or delete movies in the movie database.
Add New Movie
Mad Max Fury
Genre: Funny
Rating: 8
Release Date: 2004-07-09
Runtime: 222 mins
Budget: $20,000.00
Gross: $40,000.00
Summary: hilarious! Must watch
Details Edit Delete
Regenade Immortal
Genre: Action Thriller
Rating: 9
Release Date: 1998-06-09
Runtime: 210 mins
Budget: $6,519.00
Gross: $6,000.00
Summary: mc of this flim is very over powered and dominating.....
Details Edit Delete
Kantara
Genre: Scifi
Rating: 8
Release Date: 2023-06-09
Runtime: 190 mins
Budget: $1,900,100.00
Gross: $0.00
Summary: heyyyyyyy
Details Edit Delete
Movie2
Genre: Scifi
Rating: 9
Movie3
Genre: Action Thriller
Rating: 6
Movie4
Genre: Comedy
Rating: 8


Update Movie
Enter the updated details about the movie
Title
Regenade Immortal
Release Date
06/09/1998
Rating (0 to 10)
9
Genre
Action Thriller
Run Time (in minutes)
210
Budget (in millions)
6519
Summary
mc of this flim is very over powered and dominating.....
Gross(in millions)
6000
Update Movie
Cancel


Update Actor
Enter the details about the actor
Name
Anmol Kc
Date of Birth
06/09/1978
Country
Nepal
Height (in inches)
68
Biography
Nepal greatest actor of all time
Gender
Male
Net Worth (in millions)
4500
Update Actor
Cancel


Manage Actors
Add, view, update, or delete actors in the movie database.
+ Add New Actor
Wang Ling
Country: China
Birth Date: 1979-12-09
Height: 73 inches
Net Worth: $690,000.00
View Edit Delete
Anmol Kc
Country: Nepal
Birth Date: 1978-06-09
Height: 68 inches
Net Worth: $4,500.00
View Edit Delete
Cillian Murphy
Country: USA
Birth Date: 1978-06-29
Height: 78 inches
Net Worth: $98,001.00
View Edit Delete
Tom cruise
Country: USA
Birth Date: 1970-06-29
Height: 76 inches
Net Worth: $78,129.00
View Edit Delete
Christian Bale
Country: England
Birth Date: 1970-06-29
Height: 74 inches
Net Worth: $60,192.00
View Edit Delete


Log in
Use a local account to log in.
Email
Password
Remember me?
Log in
Forgot your password?
Register as a new user
Resend email confirmation
Use another service to log in.
There are no external authentication services configured. See this article about setting up this ASP.NET application to support logging in via external services.


Register
Create a new account.
Email
Password
Confirm Password