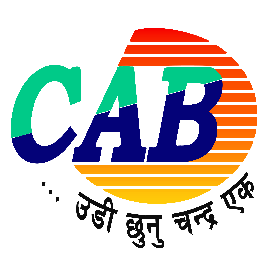
**COLLEGE OF APPLIED BUSINESS AND TECHNOLOGY**

**Gangahity, Chabahil Kathmandu**



**NET Centric Computing**

**PRACTICAL FILE-2081**

**Submitted by:**

Az Kafle (106)

College of Applied Business and Technology

B.Sc.CSIT 6th Semester

**Submitted to:**

Mr. Laxman Bhandari

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.N** | **Title** | **Signature** |
| 1 | Write a program to convert input strings from lower to upper and upper to lower case. |  |
| 2 | Write a program to create a new string from a given string where first and last characters will be interchanged. |  |
| 3 | Write a program to demonstrate the basics of class and object. |  |
| 4 | Write a program to illustrate encapsulation with properties and indexers. |  |
| 5 | Write a program that reflects the overloading and overriding of constructor and function. |  |
| 6 | Write a program to implement multiple inheritance with the use of interfaces. |  |
| 7 | Write a program to show how to handle exception in C#. |  |
| 8 | Write a program to demonstrate use of Delegate and Events |  |
| 9 | Write a program to show the use of generic classes and methods |  |
| 10 | Write a program to demonstrate the use of the method as a condition in the LINQ |  |
| 11 | Demonstrate Asynchronous programming with async, await, Task in C#. |  |
| 12 | Write a program to demonstrate dependency injection in asp.net core. |  |
| 13 | Create an ASP.NET Core application to perform CRUD operation using ADO.NET |  |
| 14 | Write a program to store and display employee information using DbContext. |  |
| 15 | Write a program to demonstrate state management server-side in asp.net core application. |  |
| 16 | Write a program to demonstrate state management client-side in asp.net core application. |  |

# Write a program to convert input strings from lower to upper and upper to lower case.

Source Code:

using System;

namespace lab

{

internal class ConvertString

{

static void Main(string[] args)

{

Console.WriteLine("Enter a string to convert to uppercase:");

string inputUpper = Console.ReadLine();

string upperCase = inputUpper.ToUpper();

Console.WriteLine("Uppercase: " + upperCase);

Console.WriteLine("Enter a string to convert to lowercase:");

string inputLower = Console.ReadLine();

string lowerCase = inputLower.ToLower();

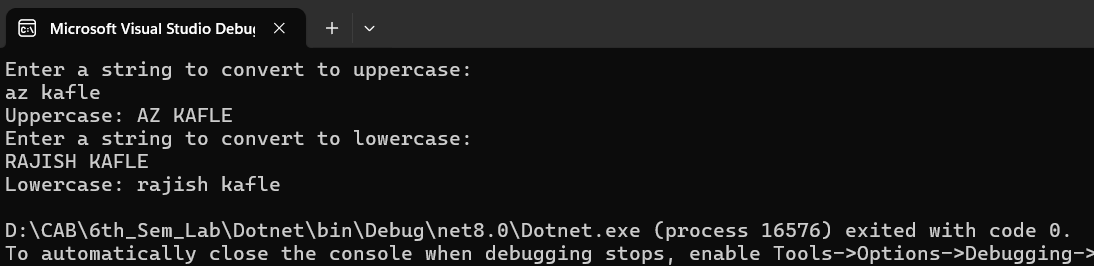
Console.WriteLine("Lowercase: " + lowerCase);

}

}

}

Output:



# Write a program to create a new string from a given string where first and last characters will be interchanged.

Source Code:

using System;

namespace lab

{

internal class StringInterchange

{

static void Main(string[] args)

{

Console.WriteLine("Enter a string:");

string input = Console.ReadLine();

string result = SwapFirstAndLastCharacters(input);

Console.WriteLine("Modified string: " + result);

}

static string SwapFirstAndLastCharacters(string input)

{

if (string.IsNullOrEmpty(input) || input.Length == 1) return input;

char[] charArray = input.ToCharArray();

char firstChar = charArray[0];

char lastChar = charArray[input.Length - 1];

charArray[0] = lastChar;

charArray[input.Length - 1] = firstChar;

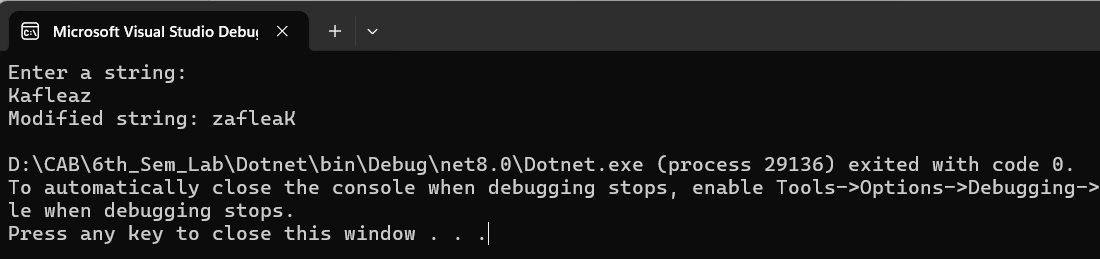
return new string(charArray);

}

}

}

Output:



# Write a program to demonstrate the basics of class and object.

Source Code:

using System;

namespace lab

{

internal class Basic\_class\_obj

{

class Person

{

public string? Name { get; set; }

public int Age { get; set; }

public void DisplayInfo()

{

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

}

class Program

{

static void Main(string[] args)

{

Person person1 = new Person();

person1.Name = "AZ";

person1.Age = 23;

Person person2 = new Person();

person2.Name = "Rajish";

person2.Age = 24;

Console.WriteLine("Person 1:");

person1.DisplayInfo();

Console.WriteLine("\nPerson 2:");

person2.DisplayInfo();

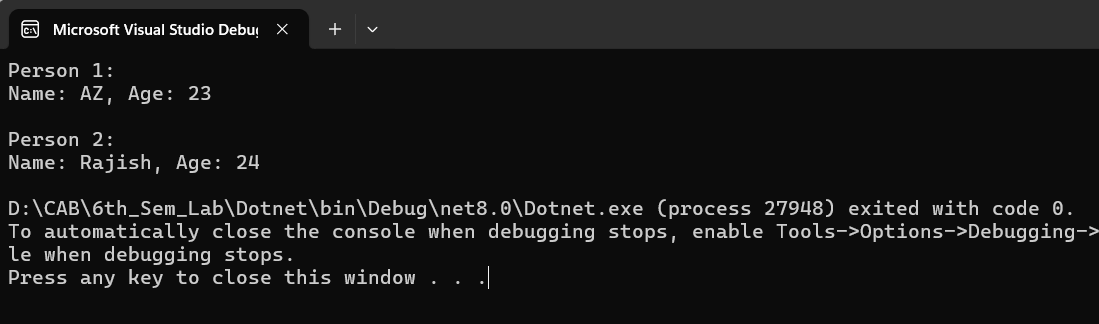
}

}

}

}

Output:



# Write a program to illustrate encapsulation with properties and indexers.

Source Code:

using System;

namespace lab

{

internal class Encapsulation

{

class Student

{

private string[] subjects = new string[5];

public string this[int index]

{

get { return subjects[index]; }

set { subjects[index] = value; }

}

public int TotalSubjects

{

get { return subjects.Length; }

}

}

class Program

{

static void Main(string[] args)

{

Student student = new Student();

student[0] = "Math";

student[1] = "Science";

student[2] = "History";

student[3] = "English";

student[4] = "Computer Science";

Console.WriteLine("Subjects:");

for (int i = 0; i < student.TotalSubjects; i++)

{

Console.WriteLine($"Subject {i + 1}: {student[i]}");

}

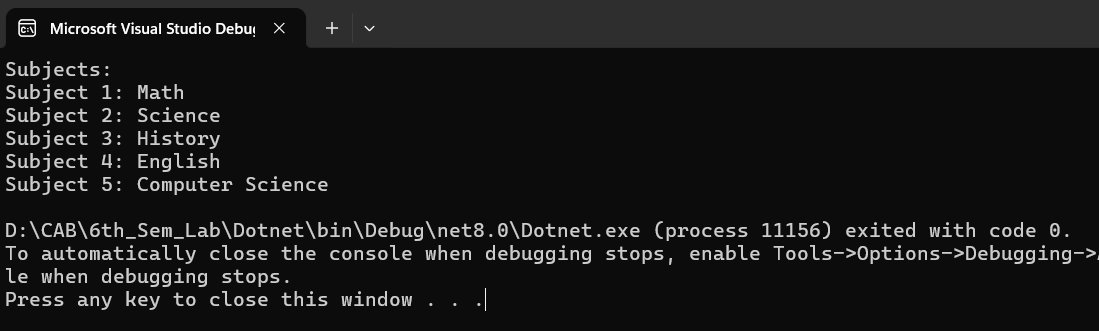
}

}

}

}

Output:



# Write a program that reflects the overloading and overriding of constructor and function.

Source Code:

using System;

namespace lab

{

internal class Overloading\_Riding

{

class Shape

{

public string Name { get; }

public Shape(string name)

{

Name = name;

}

public virtual void Display()

{

Console.WriteLine($"This is a {Name}");

}

}

class Rectangle : Shape

{

public double Width { get; }

public double Height { get; }

public Rectangle(string name, double width, double height) : base(name)

{

Width = width;

Height = height;

}

public override void Display()

{

base.Display();

Console.WriteLine($"It has width: {Width} and height: {Height}");

}

}

class Circle : Shape

{

public double Radius { get; }

public Circle(string name, double radius) : base(name)

{

Radius = radius;

}

public void Display(double area)

{

Console.WriteLine($"This is a {Name} with radius {Radius}");

Console.WriteLine($"Area: {area}");

}

}

class Program

{

static void Main(string[] args)

{

Rectangle rectangle = new Rectangle("Rectangle", 5, 10); rectangle.Display();

Circle circle = new Circle("Circle", 7);

double circleArea = CalculateCircleArea(circle.Radius);

circle.Display(circleArea);

}

static double CalculateCircleArea(double radius)

{

return Math.PI \* Math.Pow(radius, 2);

}

}

}

}

Output:



# Write a program to implement multiple inheritance with the use of interfaces.

Source Code:

using System;

namespace lab

{

internal class Multiple\_Inheritance

{

interface IShape

{

double CalculateArea();

}

interface IColor

{

string GetColor();

}

class Circle : IShape, IColor

{

private double Radius { get; }

private string Color { get; }

public Circle(double radius, string color)

{

Radius = radius;

Color = color;

}

public double CalculateArea()

{

return Math.PI \* Math.Pow(Radius, 2);

}

public string GetColor()

{

return Color;

}

}

class Program

{

static void Main(string[] args)

{

Circle redCircle = new Circle(5, "Red");

double area = redCircle.CalculateArea();

string color = redCircle.GetColor();

Console.WriteLine($"Circle Area: {area}");

Console.WriteLine($"Circle Color: {color}");

}

}

}

}

Output:



# Write a program to show how to handle exception in C#.

Source Code:

using System;

namespace lab

{

internal class ExceptionHandle

{

static void Main(string[] args)

{

try

{

Console.WriteLine("Enter a number:"); int num = int.Parse(Console.ReadLine());

int result = 10 / num;

Console.WriteLine($"Result: {result}");

}

catch (FormatException)

{

Console.WriteLine("Invalid input. Please enter a valid number.");

}

catch (DivideByZeroException)

{

Console.WriteLine("Division by zero is not allowed.");

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

finally

{

Console.WriteLine("Program execution completed.");

}

}

}

}

Output:



# Write a program to demonstrate use of Delegate and Events.

Source Code:

using System;

namespace lab

{

internal class Use\_Delegeate\_Events

{

public delegate void EventHandler(string message); class Publisher

{

public event EventHandler Notify;

public void DoSomething()

{

Console.WriteLine("Loading. . . . . ");

Notify?.Invoke("Loaded Successfully.");

}

}

class Subscriber

{

public void Subscribe(Publisher publisher)

{

publisher.Notify += HandleEvent;

}

public void Unsubscribe(Publisher publisher)

{

publisher.Notify -= HandleEvent;

}

private void HandleEvent(string message)

{

Console.WriteLine($"Event handled: {message}");

}

}

class Program

{

static void Main(string[] args)

{

Publisher publisher = new Publisher();

Subscriber subscriber = new Subscriber();

subscriber.Subscribe(publisher);

publisher.DoSomething();

subscriber.Unsubscribe(publisher);

publisher.DoSomething();

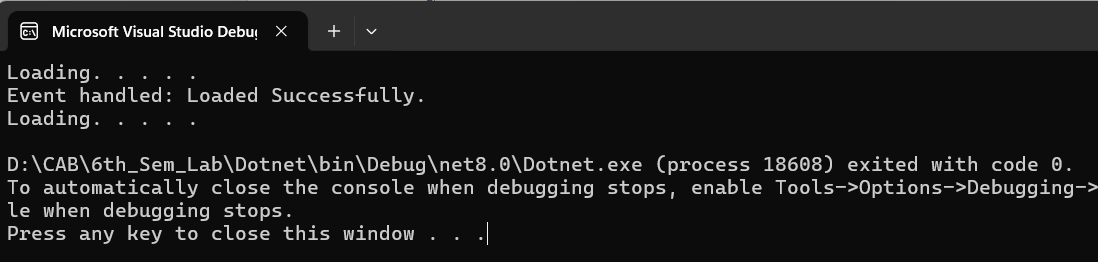
}

}

}

}

Output:



# Write a program to show the use of generic classes and methods.

Source Code:

using System;

namespace lab

{

internal class Generic\_Class

{

class Box<T>

{

private T contents; public Box(T item)

{

contents = item;

}

public T GetContents()

{

return contents;

}

}

class MathHelper

{

public static T Max<T>(T a, T b) where T : IComparable<T>

{

return a.CompareTo(b) > 0 ? a : b;

}

}

class Program

{

static void Main(string[] args)

{

Box<int> intBox = new

Box<int>(42); int intContents = intBox.GetContents();

Console.WriteLine($"Integer Contents: {intContents}");

Box<string> stringBox = new

Box<string>("Hello, Generics!");

string stringContents = stringBox.GetContents();

Console.WriteLine($"String Contents: {stringContents}");

int maxInt = MathHelper.Max(10, 20);

Console.WriteLine($"Max Integer: {maxInt}");

double maxDouble = MathHelper.Max(3.14, 2.71);

Console.WriteLine($"Max Double: {maxDouble}");

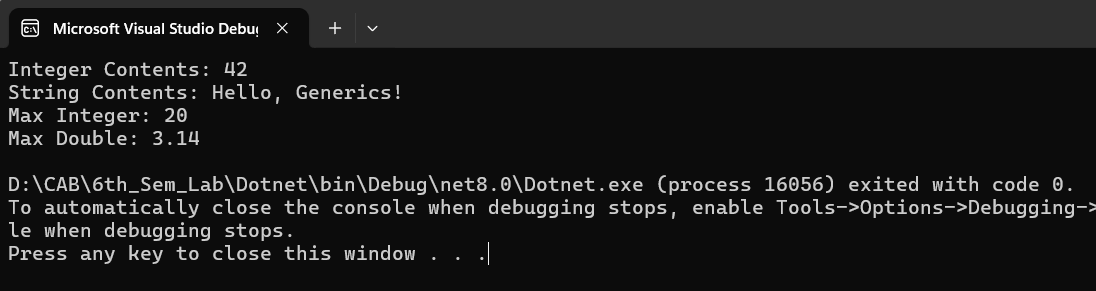
}

}

}

}

Output:



# Write a program to demonstrate the use of the method as a condition in the LINQ.

Source Code:

using System;

using System.Linq;

namespace lab

{

internal class Linq

{

class Person

{

public string Name { get; set; }

public int Age { get; set; }

}

class Program

{

static void Main(string[] args)

{

List<Person> people = new List<Person>

{

new Person { Name = "AZ", Age = 25 },

new Person { Name = "Rajish", Age = 30 },

new Person { Name = "Dev", Age = 22 }

};

var result = from person in people where IsAdult(person.Age) select person;

Console.WriteLine("Adults:");

foreach (var person in result)

{

Console.WriteLine($"Name: {person.Name}, Age: {person.Age}");

}

}

static bool IsAdult(int age)

{

return age >= 18;

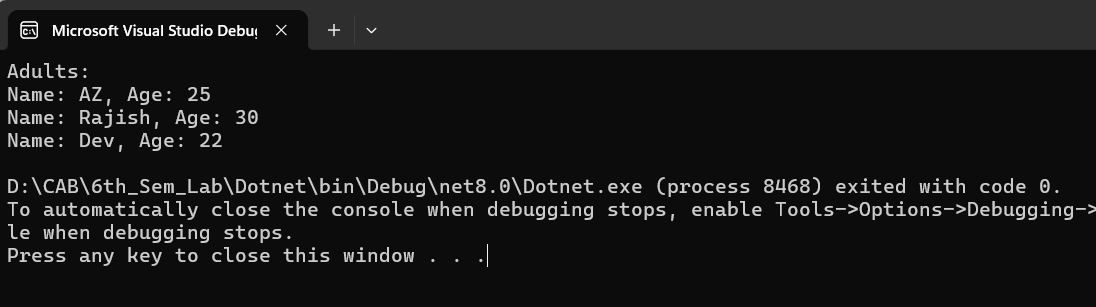
}

}

}

}

Output:



# Demonstrate Asynchronous programming with async, await, Task in C#.

Source Code:

using System;

namespace lab2

{

internal class Asynchronous

{

static void Main(string[] args)

{

Method1();

Method2();

Console.ReadKey();

}

public static async Task Method1()

{

await Task.Run(() =>

{

for (int i = 0; i < 10; i++)

{

Console.WriteLine(" Method 1");

// Do something

Task.Delay(100).Wait();

}

});

}

public static void Method2()

{

for (int i = 0; i < 5; i++)

{

Console.WriteLine(" Method 2");

// Do something

Task.Delay(100).Wait();

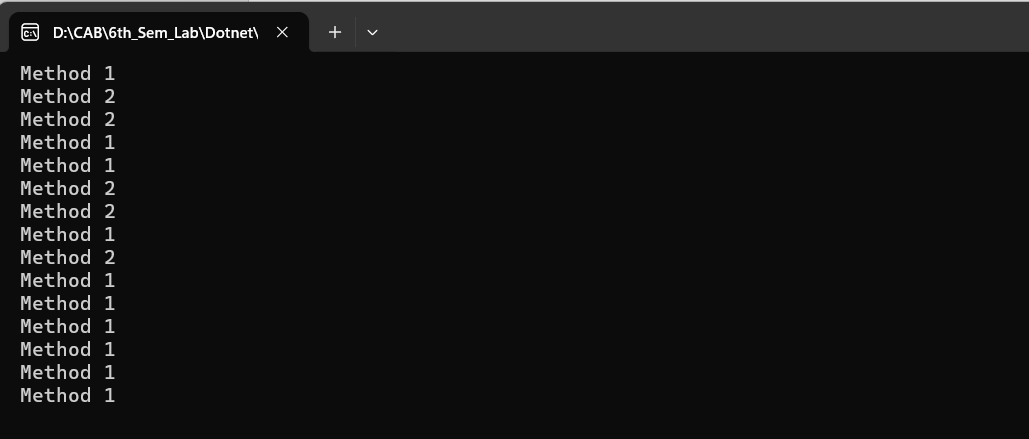
}

}

}

}

Output:



# Write a program to demonstrate dependency injection in asp.net core.

Source code:

Controllers/WeatherController.cs

using Dependency\_inj.Services;

using Microsoft.AspNetCore.Mvc;

namespace Dependency\_inj.Controllers

{

public class WeatherController : Controller

{

private readonly IWeatherService \_weatherService;

public WeatherController(IWeatherService weatherService)

{

\_weatherService = weatherService;

}

public IActionResult Index()

{

string forecast = \_weatherService.GetForecast();

return View((object)forecast);

}

}

}

Services/WeatherService.cs

using Microsoft.AspNetCore.Mvc;

namespace Dependency\_inj.Services

{

public class WeatherService : IWeatherService

{

public string GetForecast()

{

return "Today's weather is sunny!";

}

}

}

Services/IWeatherService.cs

using Microsoft.AspNetCore.Mvc;

namespace Dependency\_inj.Services

{

        public interface IWeatherService

        {

            string GetForecast();

        }

}

Index.cshtml

@model string

<h1>Weather Forecast</h1>

<p>@Model</p>

Program.cs

using Dependency\_inj.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddScoped<IWeatherService, WeatherService>();

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

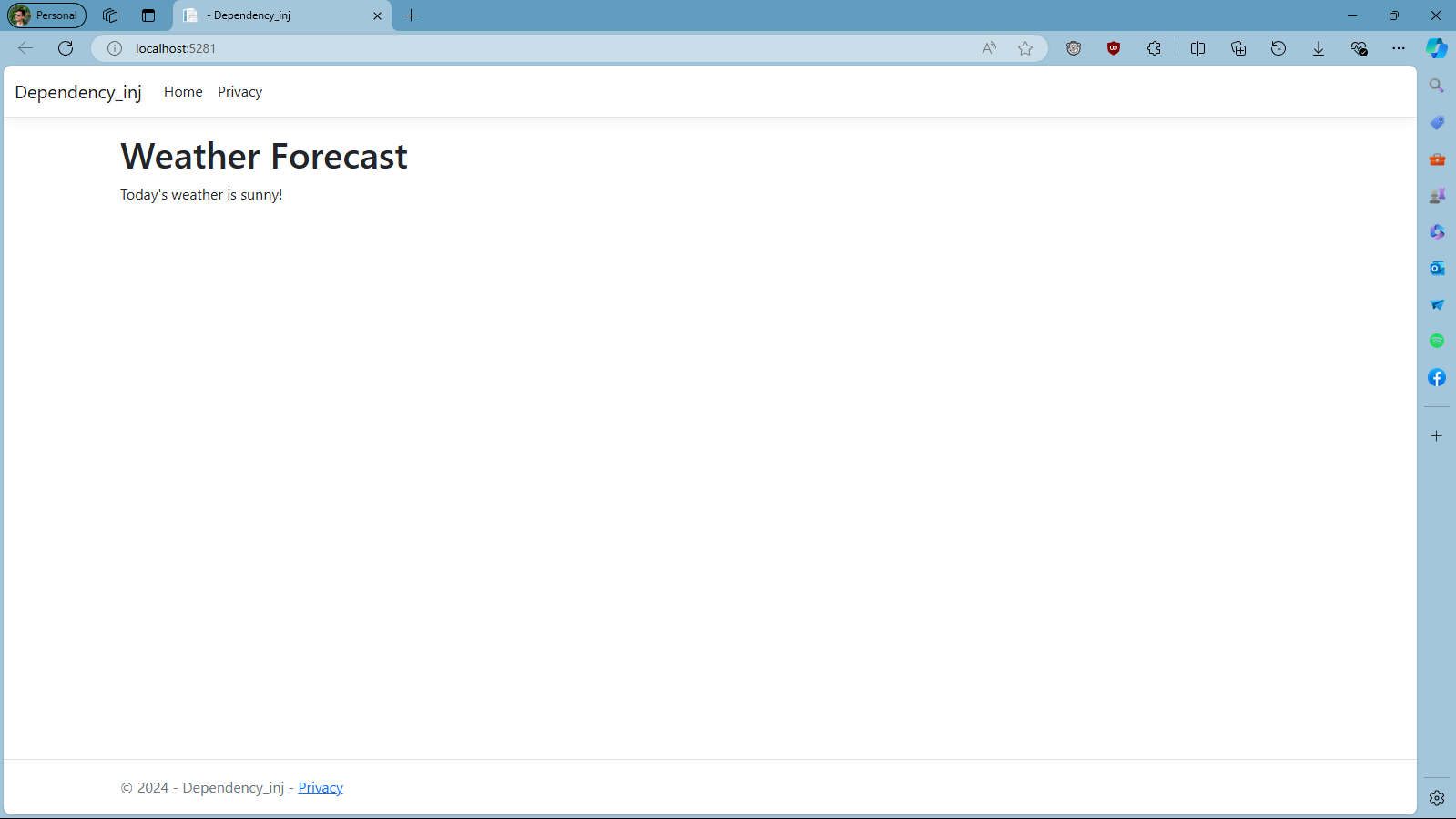
app.MapControllerRoute(

name: "default",

pattern: "{controller=Weather}/{action=Index}/{id?}");

app.Run();

Output:



# Create an ASP.NET Core application to perform CRUD operation using ADO.NET

Source code:

Controllers/ProductController.cs

using CrudOperation.Models;

using Microsoft.AspNetCore.Mvc;

using System.Data.SqlClient;

namespace CrudOperation.Controllers

{

public class ProductController : Controller

{

private string \_connectionString = "Server=(localdb)\\mssqllocaldb;Database=DotNet\_Lab;Trusted\_Connection=True;MultipleActiveResultSets=true";

public IActionResult Create()

{

return View();

}

[HttpPost]

public IActionResult Create(Product product)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string query = "INSERT INTO Products (Name, Price, Description) VALUES (@Name, @Price, @Description)";

using SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@Name", product.Name);

command.Parameters.AddWithValue("@Price", product.Price);

command.Parameters.AddWithValue("@Description", product.Description);

command.ExecuteNonQuery();

}

return RedirectToAction("Index");

}

public IActionResult Index()

{

List<Product> products = new List<Product>();

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string query = "SELECT \* FROM Products";

using SqlCommand command = new SqlCommand(query, connection);

using SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Product product = new Product

{

Id = Convert.ToInt32(reader["Id"]),

Name = reader["Name"].ToString(),

Price = Convert.ToDecimal(reader["Price"]),

Description = reader["Description"].ToString()

};

products.Add(product);

}

}

return View(products);

}

public IActionResult Edit(int id)

{

Product product = new Product();

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string query = "SELECT \* FROM Products WHERE Id = @Id";

using SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@Id", id);

using SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

product.Id = Convert.ToInt32(reader["Id"]);

product.Name = reader["Name"].ToString();

product.Price = Convert.ToDecimal(reader["Price"]);

product.Description = reader["Description"].ToString();

}

}

return View(product);

}

[HttpPost]

public IActionResult Edit(Product product)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string query = "UPDATE Products SET Name = @Name, Price = @Price, Description = @Description WHERE Id = @Id";

using SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@Id", product.Id);

command.Parameters.AddWithValue("@Name", product.Name);

command.Parameters.AddWithValue("@Price", product.Price);

command.Parameters.AddWithValue("@Description", product.Description);

command.ExecuteNonQuery();

}

return RedirectToAction("Index");

}

public IActionResult Delete(int id)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string query = "DELETE FROM Products WHERE Id = @Id";

using SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@Id", id);

command.ExecuteNonQuery();

}

return RedirectToAction("Index");

}

}

}

Models/Product.cs

namespace CrudOperation.Models

{

    public class Product

    {

        public int Id { get; set; }

        public string? Name { get; set; }

        public decimal Price { get; set; }

        public string? Description { get; set; }

    }

}

Views/Product/Create.cshtml

@model CrudOperation.Models.Product

<h2>Create Product</h2>

<form asp-action="Create">

<div class="form-group">

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

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

</div>

<div class="form-group">

<label asp-for="Price"></label>

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

</div>

<div class="form-group">

<label asp-for="Description"></label>

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

</div>

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

</form>

Views/Product/Edit.cshtml

@model CrudOperation.Models.Product

<h2>Edit Product</h2>

<form asp-action="Edit">

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

<div class="form-group">

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

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

</div>

<div class="form-group">

<label asp-for="Price"></label>

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

</div>

<div class="form-group">

<label asp-for="Description"></label>

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

</div>

<button type="submit" class="btn btn-primary">Save Changes</button>

</form>

Views/Product/Index.cshtml

@model IEnumerable<CrudOperation.Models.Product>

<h2>Product List</h2>

<p>

<a asp-action="Create" class="btn btn-primary">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Price</th>

<th>Description</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var product in Model)

{

<tr>

<td>@product.Id</td>

<td>@product.Name</td>

<td>@product.Price</td>

<td>@product.Description</td>

<td>

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

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

</td>

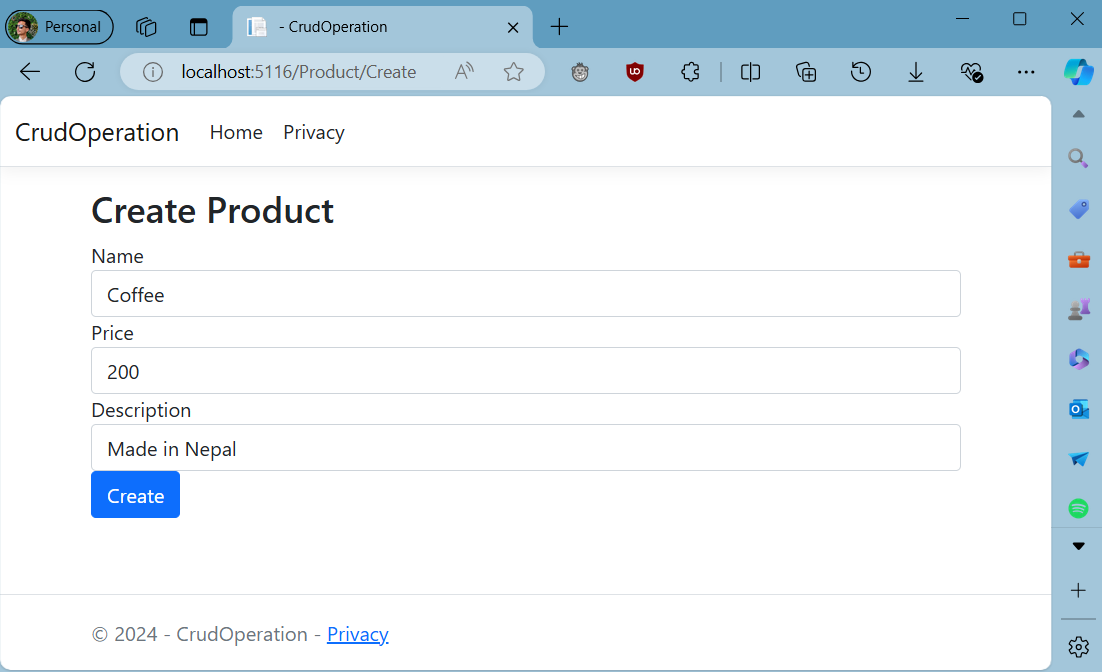
</tr>

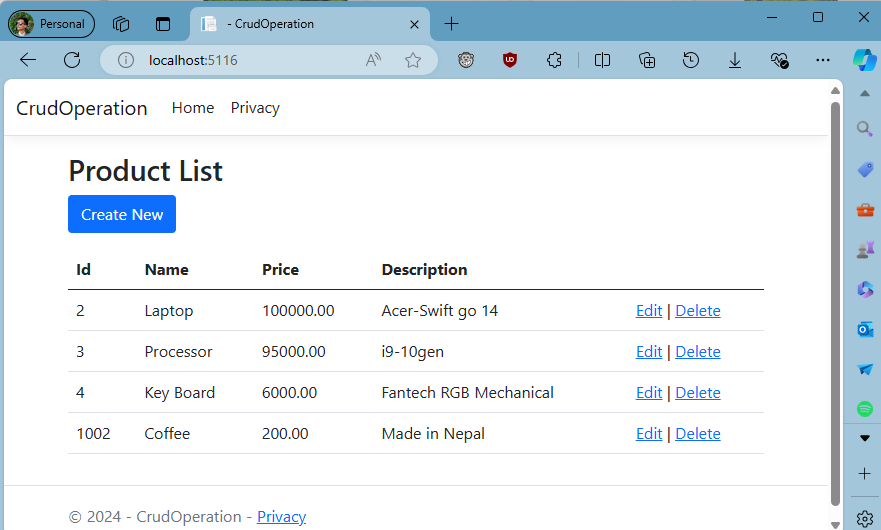
}

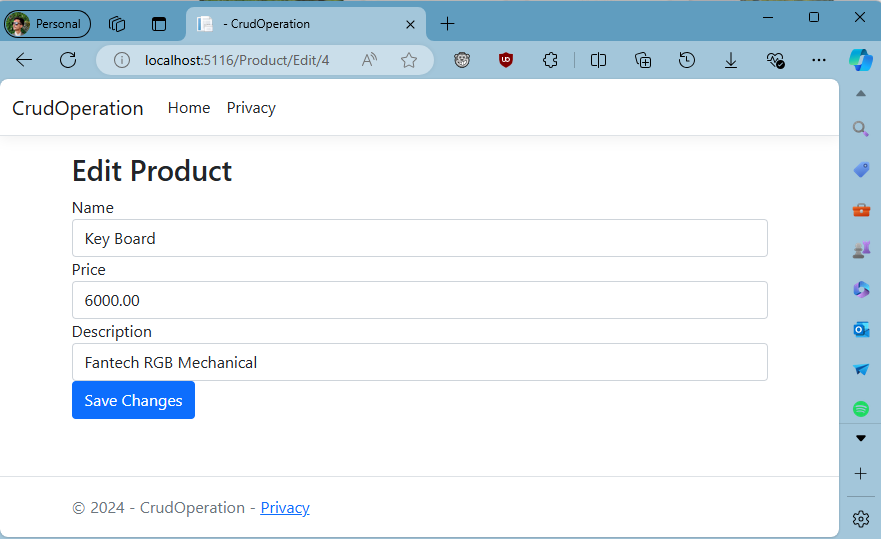
</tbody>

</table>

Output:







# Write a program to store and display employee information using DbContext.

Source code:

Controllers/HomeController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System.Collections.Generic;

using System.Linq;

using UsingDbContext.Models;

namespace PurpleStore.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationContext context;

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger, ApplicationContext \_context)

{

\_logger = logger;

context = \_context;

}

public void Add()

{

var cat1 = new Category { Name = "Toy", Description = "This is a Car Toy!" };

context.Categories.Add(cat1);

context.SaveChanges();

\_logger.LogInformation("Category added: {@cat1}", cat1);

}

public IActionResult Index()

{

Add(); // Ensure this is called to add data

List<Category> categorylist = context.Categories.ToList();

return View(categorylist);

}

}

}

Models/ApplicationContext.cs

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using UsingDbContext.Models;

namespace UsingDbContext.Models

{

    public class ApplicationContext : DbContext

    {

        private readonly IConfiguration configuration;

        public ApplicationContext(IConfiguration \_configuration)

        {

            configuration = \_configuration;

        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

            if (!optionsBuilder.IsConfigured)

            {

                optionsBuilder.UseSqlServer(configuration.GetConnectionString("test"));

            }

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

        }

        public DbSet<Category> Categories { get; set; }

    }

}

Models/Category.cs

using Microsoft.EntityFrameworkCore;

using System.ComponentModel.DataAnnotations;

namespace UsingDbContext.Models

{

public class Category

{

[Key]

public int Id { get; set; }

public string? Name { get; set; }

public string? Description { get; set; }

}

}

Views/Home/Index.cshtml

@model List<UsingDbContext.Models.Category>

<h2>Categories</h2>

<table class="table">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Description</th>

</tr>

</thead>

<tbody>

@foreach (var category in Model)

{

<tr>

<td>@category.Id</td>

<td>@category.Name</td>

<td>@category.Description</td>

</tr>

}

</tbody>

</table>

appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"test": "Server=(localdb)\\mssqllocaldb;Database=Lab\_DBcontext;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

}

Program.cs

using Microsoft.EntityFrameworkCore;

using UsingDbContext.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<ApplicationContext>(options =>

{

options.UseSqlServer(builder.Configuration.GetConnectionString("test"));

});

builder.Services.AddSession(options => { options.IdleTimeout = TimeSpan.FromMinutes(10); });

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseSession();

app.UseRouting();

app.UseAuthorization();

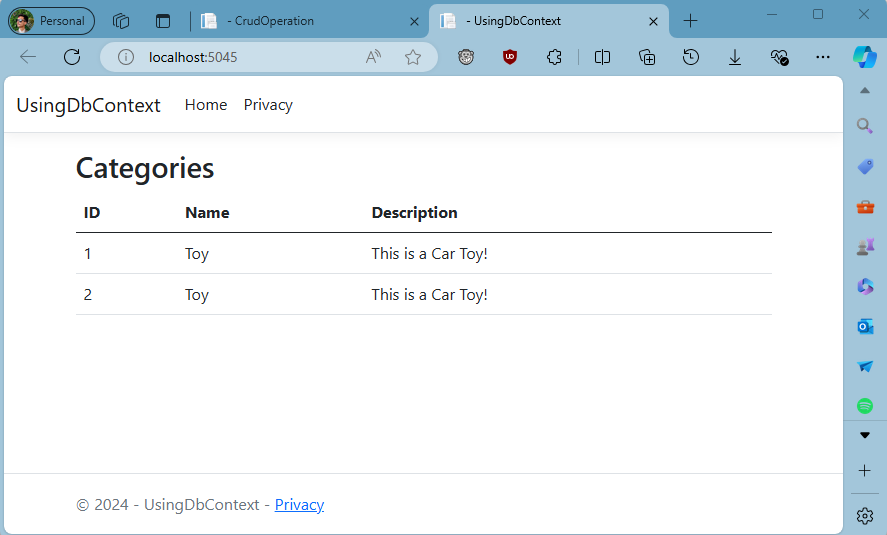
app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

Output:



# Write a program to demonstrate state management server-side in asp.net core application.

Source code:

Controller/StateController.cs

using Microsoft.AspNetCore.Mvc;

namespace Server\_Side.Controllers

{

public class StateController : Controller

{

public IActionResult Add()

{

return View();

}

[HttpPost]

public IActionResult SetUserData(string username, string message)

{

HttpContext.Session.SetString("Username", username);

TempData["Message"] = message;

return RedirectToAction("Display");

}

public IActionResult Display()

{

string username = HttpContext.Session.GetString("Username");

string message = TempData["Message"] as string;

ViewBag.Username = username; ViewBag.Message = message;

return View();

}

}

}

Views/State/Add.cshtml

@model Server\_Side.Controllers.StateController

<form method="post" asp-action="SetUserData">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br>

<label for="message">Message:</label>

<input type="text" id="message" name="message" required><br>

<button type="submit">Submit</button>

</form>

Program.cs

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

builder.Services.AddDistributedMemoryCache(); // For session state

builder.Services.AddSession(options =>

{

options.Cookie.Name = "MySessionCookie";

options.IdleTimeout = TimeSpan.FromMinutes(30);

options.Cookie.IsEssential = true;

});

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseSession();

app.UseRouting();

app.UseAuthorization();

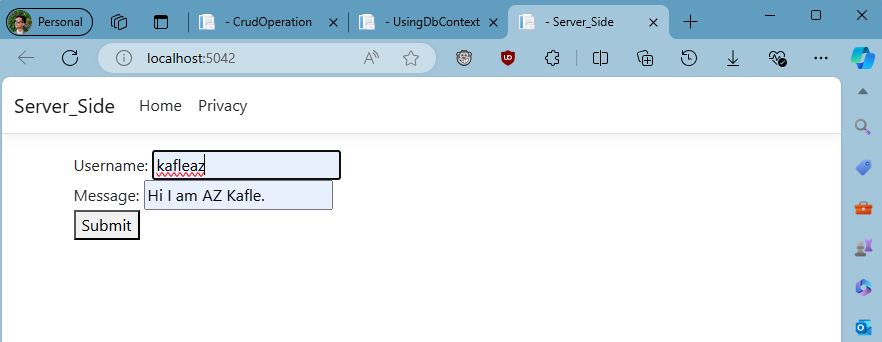
app.MapControllerRoute(

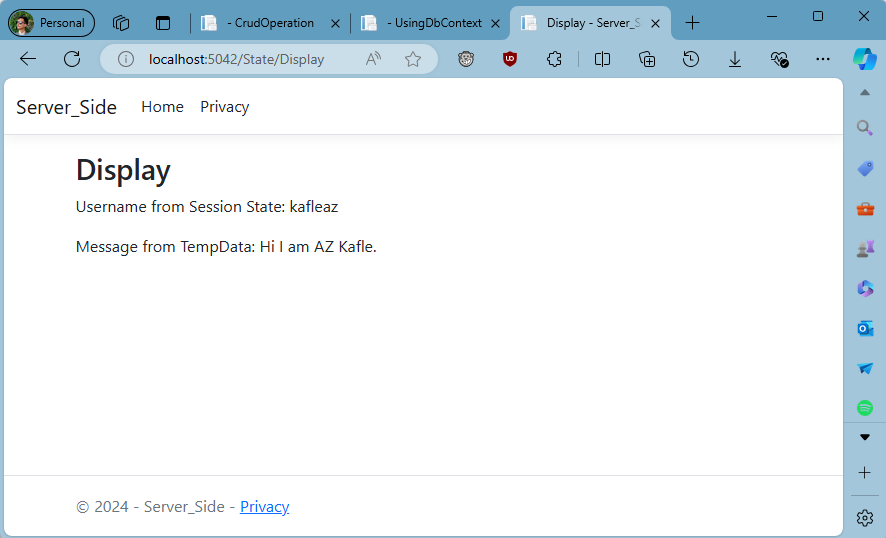
name: "default",

pattern: "{controller=State}/{action=Add}/{id?}");

app.Run();

Output:





# Write a program to demonstrate state management client-side in asp.net core application.

Source code:

Controllers/StateController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

namespace Client\_side.Controllers

{

public class StateController : Controller

{

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult SetCookie(string data)

{

// Set a cookie with the user-provided data

CookieOptions option = new CookieOptions

{

Expires = DateTime.Now.AddMinutes(30) // Cookie expiration time

};

Response.Cookies.Append("UserData", data, option);

return RedirectToAction("GetCookie"); }

public IActionResult GetCookie()

{

// Retrieve the user data from the cookie

string userData = Request.Cookies["UserData"];

ViewBag.UserData = userData;

return View();

}

}

}

Views/State/Index.cshtml

@page

@model Client\_side.Controllers.StateController

<form method="post" asp-action="SetCookie">

<label for="data">Enter data:</label>

<input type="text" name="data" required />

<button type="submit">Submit</button>

</form>

Views/State/GetCookie.cshtml

@page

@model Client\_side.Controllers.StateController

<h2>Stored User Data:</h2>

<p>@ViewBag.UserData</p>

Output:

