COLLEGE OF APPLIED BUSINESS AND TECHNOLOGY

Gangahity, Chabahil, Kathmandu-7,Nepal

(Affiliated to Tribhuvan University)

Laboratory Assignment Report of

Net Centric Computing

Submitted By:

Name: Kiran Shrestha

Roll No: 111

Semester: Sixth

Faculty: Science and Technology

Level: Bachelor

Program:CSIT

Submitted To:

Instructor: Laxman Bhandari

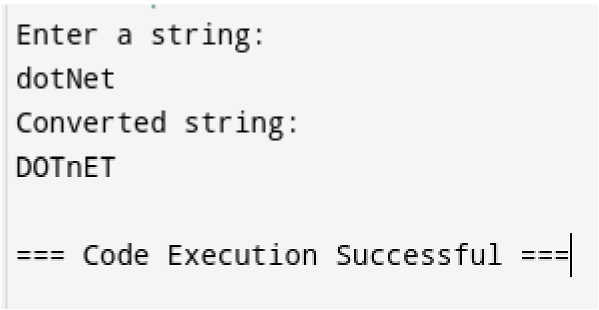
S.N.	Topics
1	Write a C# program to convert input strings from lower to upper and upper to lower case.
2	Write a C# program to create a new string from a given string where first and last characters will be interchanged.
3	Write a C# program to demonstrate the basics of class and object.
4	Write a C# program to illustrate encapsulation with properties and indexes.
5	Write a C# program that reflects the overloading and overriding of constructor and function.
6	Write a C# 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	Write a program to store and display employee information using DbContext.
14	Write a program to demonstrate state management server-side in asp.net core application.
15	Write a program to demonstrate state management client-side in asp.net core application.

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

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

```
using System;

namespace CaseConverter
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a string:");
            string input = Console.ReadLine();

            string converted = ConvertCase(input);

            Console.WriteLine("Converted string:");
            Console.WriteLine(converted);
        }

        static string ConvertCase(string input)
        {
            char[] result = new char[input.Length];

            for (int i = 0; i < input.Length; i++)
            {
                char c = input[i];
                if (char.IsLower(c))
                {
                    result[i] = char.ToUpper(c);
                }
                else if (char.IsUpper(c))
                {
                    result[i] = char.ToLower(c);
                }
                else
                {
                    result[i] = c;
                }
            }

            return new string(result);
        }
    }
}

}
```

Output:

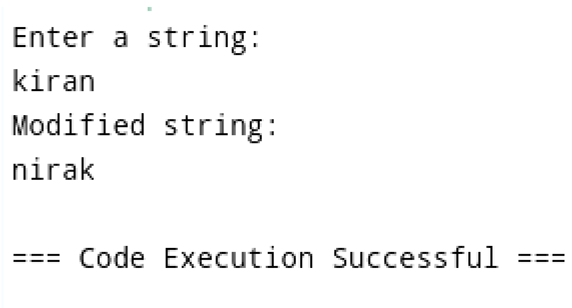


```
Enter a string:
dotNet
Converted string:
DOTnET

=== Code Execution Successful ===
```

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

```
using System;

namespace StringInterchange
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a string:");
            string input = Console.ReadLine();
                        string modifiedString = InterchangeFirstAndLast(input);
            Console.WriteLine("Modified string:");
            Console.WriteLine(modifiedString);
        }
        static string InterchangeFirstAndLast(string input)
        {
            if (string.IsNullOrEmpty(input) || input.Length == 1)
            {
                return input;
            }
            char firstChar = input[0];
            char lastChar = input[input.Length - 1];
            char[] charArray = input.ToCharArray();
            charArray[0] = lastChar;
            charArray[charArray.Length - 1] = firstChar;
            return new string(charArray);
        }
    }
}
```

Output:

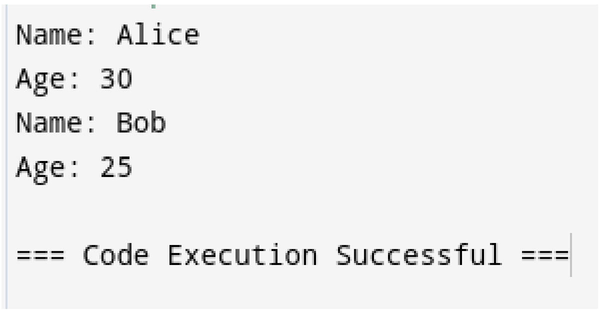


```
Enter a string:
kiran
Modified string:
nirak

=== Code Execution Successful ===
```

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

```
using System;

namespace ClassAndObjectDemo
{
    class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public Person(string name, int age)
        {
            Name = name;
            Age = age;
        }
        public void DisplayDetails()
        {
            Console.WriteLine($"Name: {Name}");
            Console.WriteLine($"Age: {Age}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Person person1 = new Person("Alice", 30);
            person1.DisplayDetails();
            Person person2 = new Person("Bob", 25);
            person2.DisplayDetails();
        }
    }
}
```

Output:

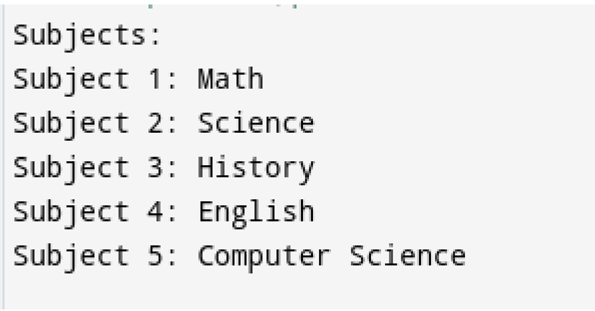


```
Name: Alice
Age: 30
Name: Bob
Age: 25

=== Code Execution Successful ===
```

4) Write a program to illustrate encapsulation with properties and indexers.

```
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:



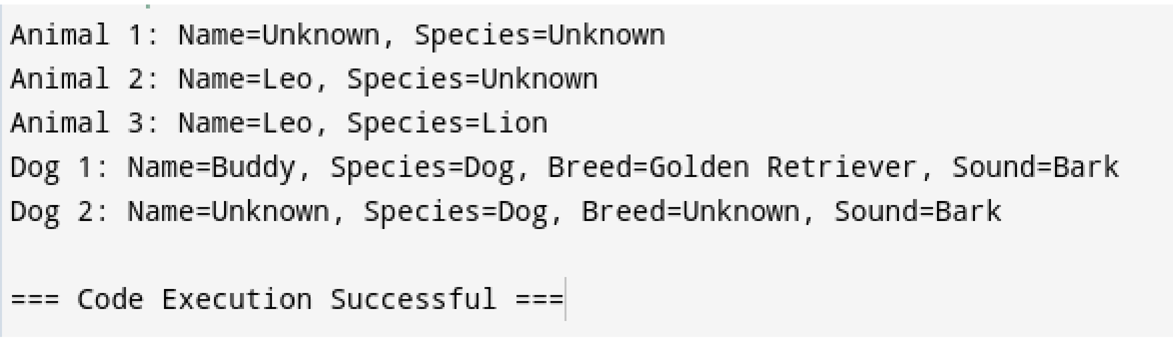
```
Subjects:
Subject 1: Math
Subject 2: Science
Subject 3: History
Subject 4: English
Subject 5: Computer Science
```

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

```
using System;

// Base class
class Animal
{
    public string Name { get; set; }
    public string Species { get; set; }

    // Constructor with default parameters
    public Animal(string name = "Unknown", string species = "Unknown")
    {
        Name = name;
        Species = species;
    }

    // Method to make sound (to be overridden)
    public virtual string MakeSound()
    {
        return "Some generic sound";
    }
}

// Derived class
class Dog : Animal
{
    public string Breed { get; set; }

    // Constructor overloading
    public Dog(string name = "Unknown", string species = "Dog", string breed = "Unknown")
        : base(name, species)
    {
        Breed = breed;
    }

    // Method overriding
    public override string MakeSound()
    {
        return "Bark";
    }
}
```

```
class Program
{
    static void Main()
    {
        // Demonstrate constructor overloading
        Animal animal1 = new Animal();
        Animal animal2 = new Animal("Leo");
        Animal animal3 = new Animal("Leo", "Lion");

        Console.WriteLine($"Animal 1: Name={animal1.Name},
Species={animal1.Species}");
        Console.WriteLine($"Animal 2: Name={animal2.Name},
Species={animal2.Species}");
        Console.WriteLine($"Animal 3: Name={animal3.Name},
Species={animal3.Species}");

        // Demonstrate method overriding
        Dog dog1 = new Dog("Buddy", breed: "Golden Retriever");
        Dog dog2 = new Dog();

        Console.WriteLine($"Dog 1: Name={dog1.Name}, Species={dog1.Species},
Breed={dog1.Breed}, Sound={dog1.MakeSound()}");
        Console.WriteLine($"Dog 2: Name={dog2.Name}, Species={dog2.Species},
Breed={dog2.Breed}, Sound={dog2.MakeSound()}");
    }
}
```

Output:

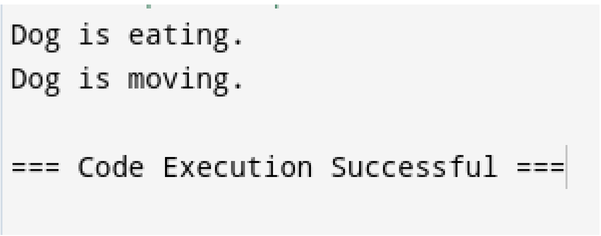


```
Animal 1: Name=Unknown, Species=Unknown
Animal 2: Name=Leo, Species=Unknown
Animal 3: Name=Leo, Species=Lion
Dog 1: Name=Buddy, Species=Dog, Breed=Golden Retriever, Sound=Bark
Dog 2: Name=Unknown, Species=Dog, Breed=Unknown, Sound=Bark

=== Code Execution Successful ===
```

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

```
using System;

// Define the first interface
public interface IAnimal
{
   void Eat();
}

// Define the second interface
public interface IMovable
{
   void Move();
}

// Implement the interfaces in a class
public class Dog : IAnimal, IMovable
{
   public void Eat()
   {
      Console.WriteLine("Dog is eating.");
   }

   public void Move()
   {
      Console.WriteLine("Dog is moving.");
   }
}

class Program
{
   static void Main()
   {
      // Create an instance of Dog
      Dog dog = new Dog();

      // Call methods from interfaces
      dog.Eat();
      dog.Move();
   }
}
```

Output:

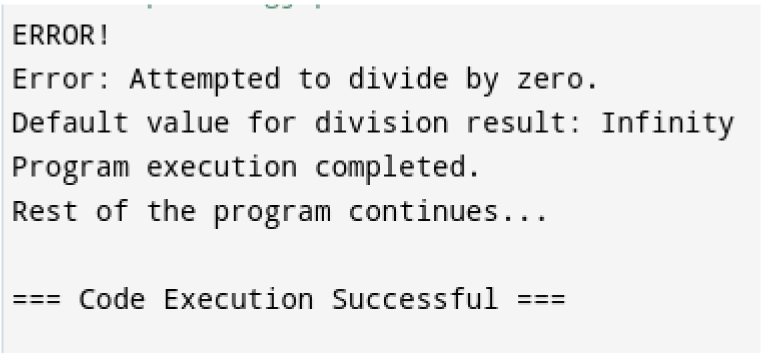


```
Dog is eating.
Dog is moving.

=== Code Execution Successful ===
```

7) Write a program to show how to handle exception in C#

```
using System;

class Program
{
    static void Main()
    {
        try
        {
            // Example: Divide by zero exception
            int numerator = 10;
            int denominator = 0;
            int result = numerator / denominator;  // This line will throw an exception
            Console.WriteLine($"Result of division: {result}");
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
            // Handle the exception (e.g., provide a default value)
            Console.WriteLine("Default value for division result: Infinity");
        }
        catch (Exception ex)
        {
            // Catch-all block for any other exceptions
            Console.WriteLine($"Unexpected error occurred: {ex.Message}");
        }
        finally
        {
            // Optional finally block, executes whether an exception occurred or not
            Console.WriteLine("Program execution completed.");
        }

        Console.WriteLine("Rest of the program continues...");
    }
}
```

Output:

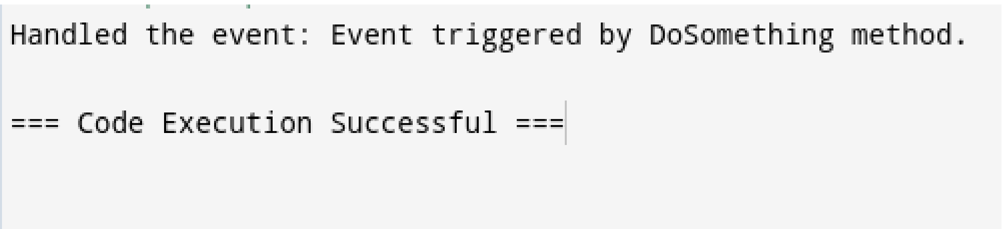


```
ERROR!
Error: Attempted to divide by zero.
Default value for division result: Infinity
Program execution completed.
Rest of the program continues...

=== Code Execution Successful ===
```

8) Write a program to demonstrate use of Delegate and Events.

```
using System;

// Step 1: Define a delegate
public delegate void EventHandler(string message);

// Step 2: Define a class that contains an event
public class EventPublisher
{
    // Step 3: Define an event based on the delegate
    public event EventHandler RaiseCustomEvent;

    // Step 4: Method to raise the event
    public void DoSomething()
    {
        // Step 5: Raise the event
        OnRaiseCustomEvent("Event triggered by DoSomething method.");
    }

    // Step 6: Method to invoke the event
    protected virtual void OnRaiseCustomEvent(string message)
    {
        RaiseCustomEvent?.Invoke(message);  // Invoke the event
    }
}

// Step 7: Define a class that subscribes to the event
public class EventSubscriber
{
    // Step 8: Event handler method
    public void HandleCustomEvent(string message)
    {
        Console.WriteLine($"Handled the event: {message}");
    }
}

class Program
{
    static void Main()
    {
        // Step 9: Create instances of publisher and subscriber
        EventPublisher publisher = new EventPublisher();
        EventSubscriber subscriber = new EventSubscriber();

        // Step 10: Subscribe to the event
        publisher.RaiseCustomEvent += subscriber.HandleCustomEvent;

        // Step 11: Trigger the event
        publisher.DoSomething();

        // Step 12: Unsubscribe from the event (optional)
        publisher.RaiseCustomEvent -= subscriber.HandleCustomEvent;
    }
}
```

Output:



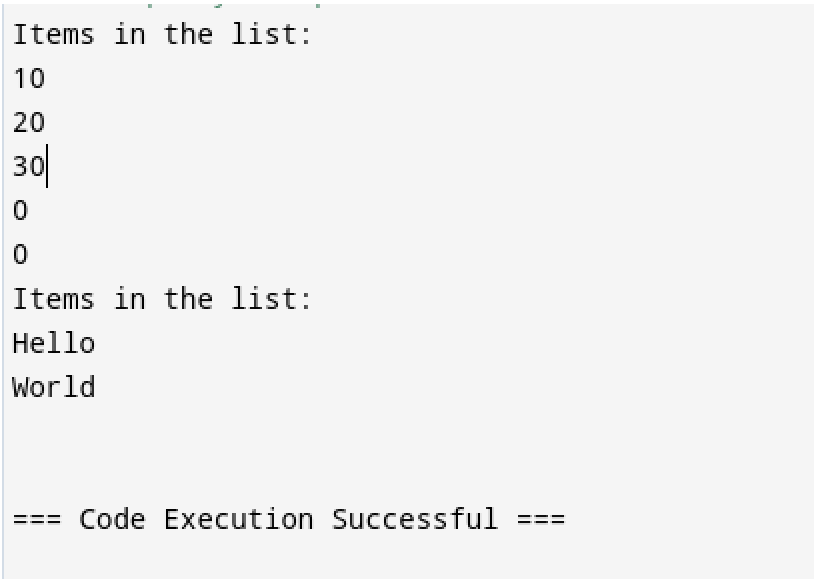
```
Handled the event: Event triggered by DoSomething method.

=== Code Execution Successful ===
```

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

```
using System;

// Generic class
public class GenericList<T>
{
   private T[] _items;
   private int _currentIndex;

   // Constructor
   public GenericList(int capacity)
   {
      _items = new T[capacity];
      _currentIndex = 0;
   }

   // Method to add an item to the list
   public void Add(T item)
   {
      if (_currentIndex < _items.Length)
      {
         _items[_currentIndex] = item;
         _currentIndex++;
      }
      else
      {
         Console.WriteLine("List is full. Cannot add more items.");
      }
   }

   // Method to display all items in the list
   public void DisplayItems()
   {
      Console.WriteLine("Items in the list:");
      foreach (var item in _items)
      {
         Console.WriteLine(item);
      }
   }
}

class Program
{
   static void Main()
   {
      // Creating a list of integers
```

```
        GenericList<int> intList = new GenericList<int>(5);
        intList.Add(10);
        intList.Add(20);
        intList.Add(30);
        intList.DisplayItems();

        // Creating a list of strings
        GenericList<string> stringList = new GenericList<string>(3);
        stringList.Add("Hello");
        stringList.Add("World");
        stringList.DisplayItems();
    }
}
```

Output:

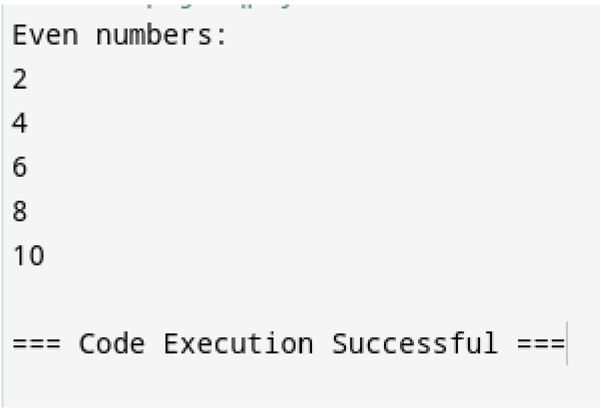


```
Items in the list:
10
20
30
0
0
Items in the list:
Hello
World


=== Code Execution Successful ===
```

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

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

class Program
{
   static void Main()
   {
      // Sample list of integers
      List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

      // Using LINQ to filter even numbers
      IEnumerable<int> evenNumbers = numbers.Where(IsEven);

      // Display the filtered numbers
      Console.WriteLine("Even numbers:");
      foreach (var number in evenNumbers)
      {
         Console.WriteLine(number);
      }
   }

   // Method to check if a number is even
   static bool IsEven(int number)
   {
      return number % 2 == 0;
   }
}
```

Output:

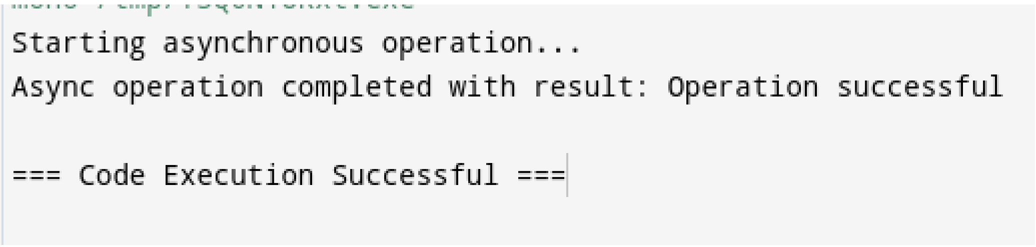


```
Even numbers:
2
4
6
8
10

=== Code Execution Successful ===
```

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

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        Console.WriteLine("Starting asynchronous operation...");

        try
        {
            // Call an asynchronous method and await the result
            string result = await SimulateAsyncOperation();

            // Display the result
            Console.WriteLine($"Async operation completed with result: {result}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }

    // Asynchronous method to simulate work
    static async Task<string> SimulateAsyncOperation()
    {
        await Task.Delay(2000); // Simulate a delay of 2 seconds (2000 milliseconds)
        return "Operation successful";
    }
}
```

Output:



```
Starting asynchronous operation...
Async operation completed with result: Operation successful

=== Code Execution Successful ===
```

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

Step1: Create a asp.net core MVC applications

Step2: Define a Service Interface and Implementation

- Create a new folder “Services”
- Create a new file “MessageService.cs”

```
public interface IMessageService
{
   string GetMessage();
}

public class WelcomeMessageService : IMessageService
{
   public string GetMessage()
   {
      return "Hello from WelcomeMessageService!";
   }
}
```

Step3: Configure Dependency Injection in Program.cs
Add this line of code above “var app = builder.Build();”

```
builder.Services.AddScoped<IMessageService, WelcomeMessageService>();
```

Step4: Use Dependency Injection in a Controller

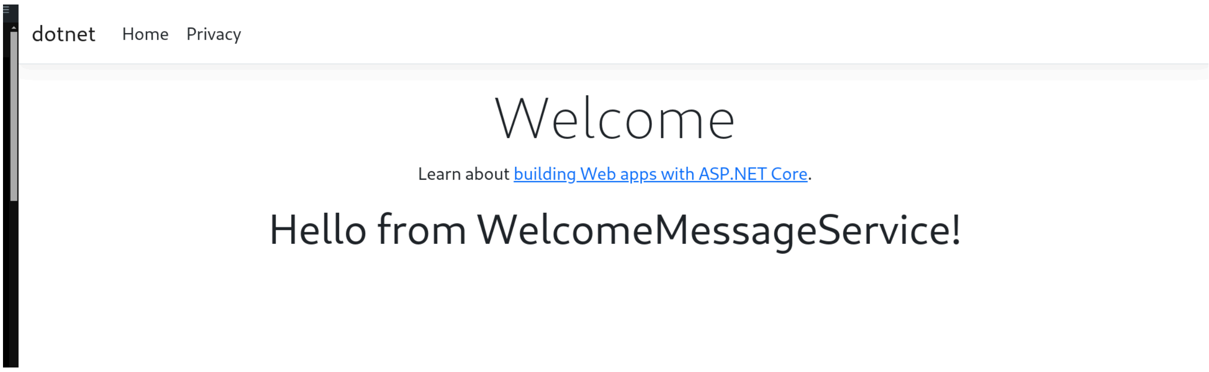
```
public class HomeController : Controller
{
   private readonly IMessageService _services;

   public HomeController(IMessageService services)
   {
      _services = services;
   }

   public IActionResult Index()
   {
      string message = _services.GetMessage();
      return View(model: message);
   }
}
```

Step5: Create a View to Display the Message

Inside Index.cshtml write the following code:

```
@{
    ViewData["Title"] = "Home Page";
}
@model string

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building
Web apps with ASP.NET Core</a>.</p>
    <h1>@Model</h1>
</div>
```

Output:



Welcome

Learn about building Web apps with ASP.NET Core.

Hello from WelcomeMessageService!

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

First, make sure you have installed the necessary packages. If you're using .NET Core, you can add these packages via the NuGet Package Manager:

```
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
```
￼

Check dotnet.csproj to check if these packages has been installed in your project.

Step1: Create Employee model

```
using System.ComponentModel.DataAnnotations;
namespace EmployeeManagement.Models
{
   public class Employee
   {
      [Key]
      public int Id { get; set; }
      public string? Name { get; set; }
      public string? Position { get; set; }
      public decimal Salary { get; set; }
   }
}
```

Step2: DbContext Class

```
using Microsoft.EntityFrameworkCore;
using EmployeeManagement.Models;
namespace EmployeeApp.Data
{
   public class ApplicationDbContext : DbContext
   {
      public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
         : base(options)
      {
      }

      public DbSet<Employee>? Employees { get; set; }
   }
}
```

Step3: Create Database migration

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Step4: Create <u>EmployeeController.cs</u>

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;
using EmployeeApp.Data;
using EmployeeManagement.Models;

public class EmployeeController : Controller
{
   private readonly ApplicationDbContext _context;

   public EmployeeController(ApplicationDbContext context)
   {
      _context = context;
   }

   public async Task<IActionResult> Index()
   {
      return View(await _context.Employees.ToListAsync());
   }

   public IActionResult Create()
   {
      return View();
   }

   [HttpPost]
   [ValidateAntiForgeryToken]
   public async Task<IActionResult> Create([Bind("Id,Name,Position,Salary")]
Employee employee)
   {
      if (ModelState.IsValid)
      {
         _context.Add(employee);
         await _context.SaveChangesAsync();
         return RedirectToAction(nameof(Index));
      }
```

```
            return View(employee);
        }
    }
```

Step 5: Configure Program.cs

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
{

options.UseSqlite(builder.Configuration.GetConnectionString("SQLiteConnection
"));
    options.EnableSensitiveDataLogging(); // Enable if needed for debugging

});
```

Step 6: Configure appsettings.json

```
"ConnectionStrings": {
   "SQLiteConnection": "Data Source=mydatabase.db"
 }
```

Step 6: Create Index.cshtml and Create.cshtml files inside “Views>Employee” folder

Index.cshtml

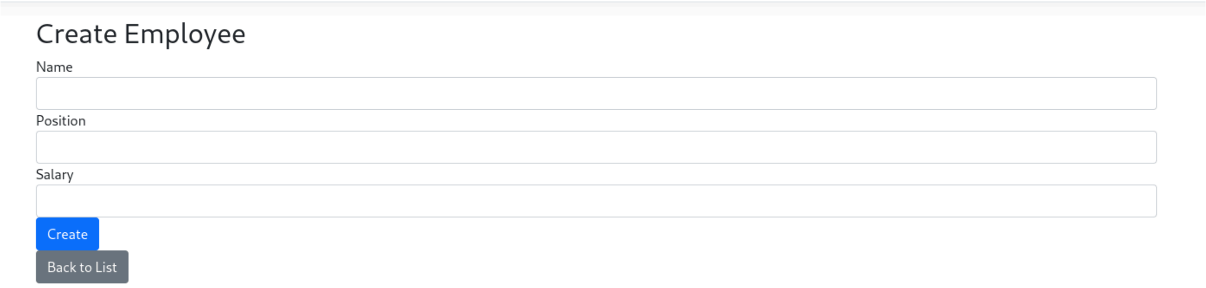
```
@model IEnumerable<EmployeeManagement.Models.Employee>

<h2>Employee List</h2>
<table class="table">
  <thead>
    <tr>
      <th>Name</th>
      <th>Position</th>
      <th>Salary</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in Model)
    {
      <tr>
        <td>@item.Name</td>
        <td>@item.Position</td>
        <td>@item.Salary</td>
      </tr>
    }
  </tbody>
</table>
```

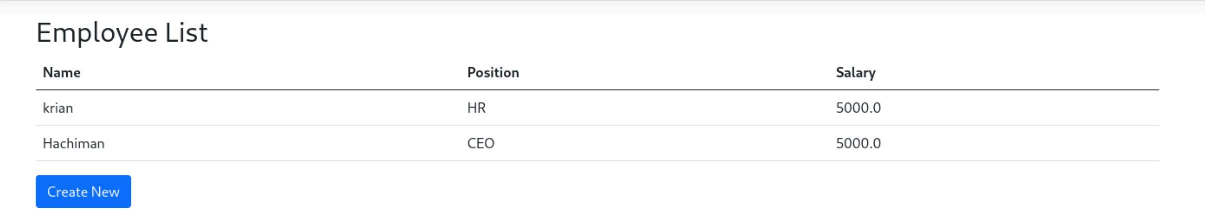
```
<a href="@Url.Action("Create")" class="btn btn-primary">Create New</a>
```

Create.cshtml

```
@model EmployeeManagement.Models.Employee

<h2>Create Employee</h2>

<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="Position" class="control-label"></label>
        <input asp-for="Position" class="form-control" />
        <span asp-validation-for="Position" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Salary" class="control-label"></label>
        <input asp-for="Salary" class="form-control" />
        <span asp-validation-for="Salary" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
<a asp-action="Index" class="btn btn-secondary">Back to List</a>
```

Output:





Employee List

Name	Position	Salary
krian	HR	5000.0
Hachiman	CEO	5000.0

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

Step 1: Create a controller "StateController.cs"

```
using Microsoft.AspNetCore.Mvc;
namespace State.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();
    }
  }
}
```

Step 2: Configure Program.cs file to use session

```
builder.Services.AddDistributedMemoryCache(); // For session state
builder.Services.AddSession(options =>
{
options.Cookie.Name = "MySessionCookie";
options.IdleTimeout =
System.TimeSpan.FromMinutes(30);
options.Cookie.IsEssential = true;
});
```

Then insert "app.UseSession();";

```
app.UseRouting();
```

Step 3: Create Add.cshtml and Display.cshtml file inside Views>State folder

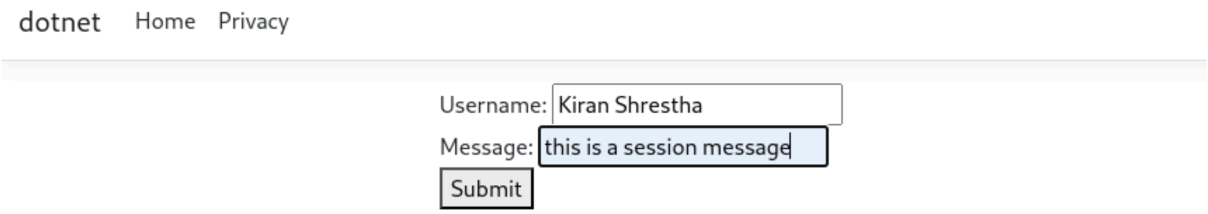
Add.cshtml

```
@model State.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>
```

Display.cshtml

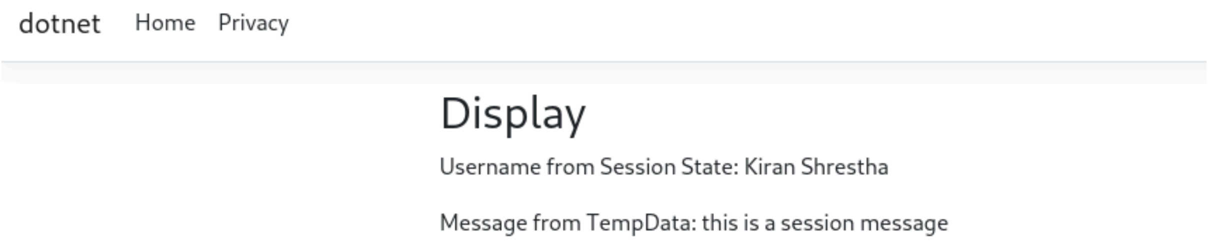
```
@{
   ViewData["Title"] = "Display";
}
<h2>Display</h2>
<div>
   <p>Username from Session State: @ViewBag.Username</p>
   <p>Message from TempData: @ViewBag.Message</p>
</div>
```

Output:



Username: Kiran Shrestha

Message: this is a session message



Display

Username from Session State: Kiran Shrestha

Message from TempData: this is a session message

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

Step 1: Create a controller “StateController.cs”

```
using Microsoft.AspNetCore.Mvc;
namespace State.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();
              option.Expires = DateTime.Now.AddMinutes(30);
              Response.Cookies.Append("UserData", data, option);
              return RedirectToAction("Index");
            }
            public IActionResult GetCookie()
            {
              // Retrieve the user data from the cookie
              string userData = Request.Cookies["UserData"];
              ViewBag.UserData = userData;
              return View();
            }
  }
}
```

Step 2: Create Index.cshtml and GetCookie.cshtml files inside View>State folder.

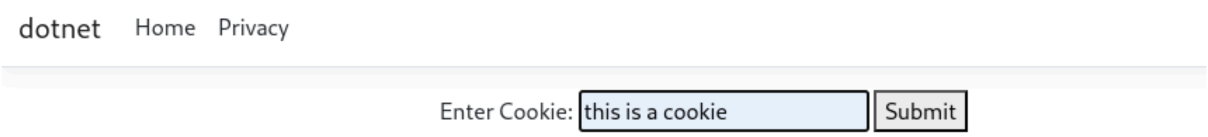
Index.cshtml

```
@page
@model State.Controllers.StateController

<form method="post" asp-action="SetCookie">
  <label for="data">Enter Cookie:</label>
  <input type="text" name="data" required />
  <button type="submit">Submit</button>
</form>
```

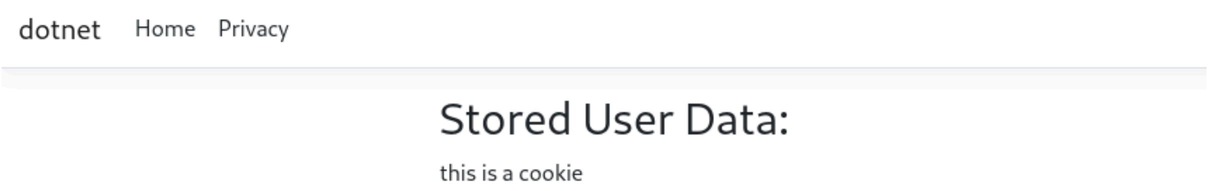
GetCookie.cshtml

```
@page
@model State.Controllers.StateController

<h2>Stored User Data:</h2>
<p>@ViewBag.UserData</p>
```

Output:



Enter Cookie: this is a cookie



Stored User Data:

this is a cookie

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

Step1: Install the required package

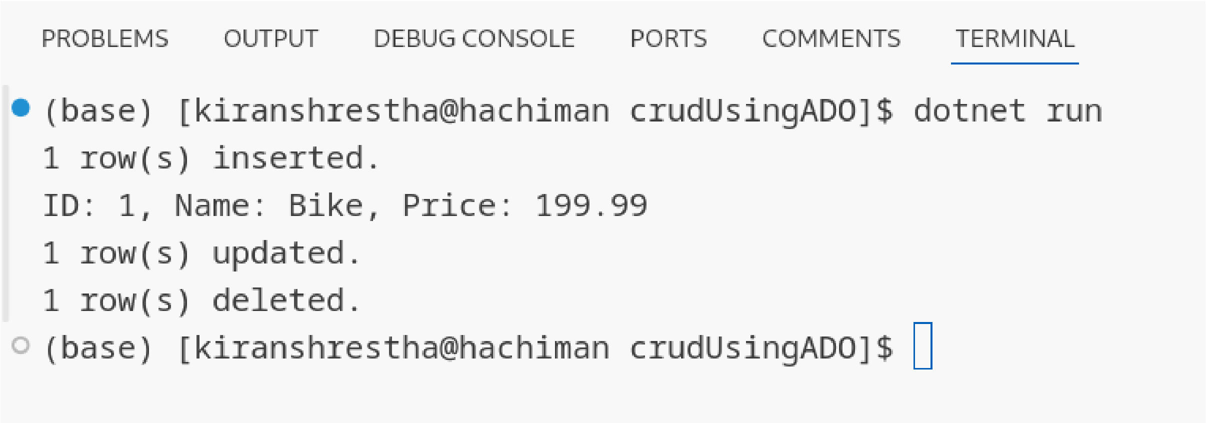
```
dotnet add package Microsoft.Data.Sqlite
```

Step2: Write the following code in your Program.cs file

```
using System;
using Microsoft.Data.Sqlite;

class Program
{
   private static string connectionString = "Data Source=products.db";

   static void Main()
   {
      CreateTable(); // Ensure the table is created
      // Example usage
      CreateProduct("Bike", 199.99m);
      ReadProducts();
      UpdateProduct(1, "Mountain Bike", 299.99m);
      DeleteProduct(1);
   }

   static void CreateTable()
   {
      using (var connection = new SqliteConnection(connectionString))
      {
         connection.Open();
         string createTableQuery = @"CREATE TABLE IF NOT EXISTS Products (
                              Id INTEGER PRIMARY KEY AUTOINCREMENT,
                              Name TEXT NOT NULL,
                              Price REAL NOT NULL
                            );";
         using (var command = new SqliteCommand(createTableQuery, connection))
         {
            command.ExecuteNonQuery();
         }
      }
   }

   static void CreateProduct(string name, decimal price)
   {
      using (var connection = new SqliteConnection(connectionString))
```

```
        {
            connection.Open();
            string sql = "INSERT INTO Products (Name, Price) VALUES (@Name,
@Price)";
            using (var command = new SqliteCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@Name", name);
                command.Parameters.AddWithValue("@Price", price);
                int rowsAffected = command.ExecuteNonQuery();
                Console.WriteLine($"{rowsAffected} row(s) inserted.");
            }
        }
    }

    static void ReadProducts()
    {
        using (var connection = new SqliteConnection(connectionString))
        {
            connection.Open();
            string sql = "SELECT * FROM Products";
            using (var command = new SqliteCommand(sql, connection))
            {
                using (var reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        Console.WriteLine($"ID: {reader["Id"]}, Name: {reader["Name"]}, Price:
{reader["Price"]}");
                    }
                }
            }
        }
    }

    static void UpdateProduct(int id, string name, decimal price)
    {
        using (var connection = new SqliteConnection(connectionString))
        {
            connection.Open();
            string sql = "UPDATE Products SET Name = @Name, Price = @Price WHERE
Id = @Id";
            using (var command = new SqliteCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@Id", id);
                command.Parameters.AddWithValue("@Name", name);
                command.Parameters.AddWithValue("@Price", price);
                int rowsAffected = command.ExecuteNonQuery();
                Console.WriteLine($"{rowsAffected} row(s) updated.");
            }
        }
    }

    static void DeleteProduct(int id)
    {
        using (var connection = new SqliteConnection(connectionString))
        {
            connection.Open();
            string sql = "DELETE FROM Products WHERE Id = @Id";
            using (var command = new SqliteCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@Id", id);
                int rowsAffected = command.ExecuteNonQuery();
                Console.WriteLine($"{rowsAffected} row(s) deleted.");
            }
        }
    }
}
```

Output:



```
(base) [kiranshrestha@hachiman crudUsingADO]$ dotnet run
1 row(s) inserted.
ID: 1, Name: Bike, Price: 199.99
1 row(s) updated.
1 row(s) deleted.
(base) [kiranshrestha@hachiman crudUsingADO]$ 
```