

**COLLEGE OF APPLIED BUSINESS AND TECHNOLOGY**  
**Chabahil, Gangahity, Kathmandu**



**NET Centric Computing**  
**PRACTICAL EXAM-2024**



**Submitted by:**

Arnav Sharma (103)

College of Applied Business and Technology

B.Sc.CSIT 6<sup>th</sup> Semester

**Submitted to:**

Mr. Laxman Bhandari

## Experiment 1

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

### Code

```
using System;
class ConsoleApp
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter a string:");
        string? input = Console.ReadLine();
        if (input != null)
        {
            Console.WriteLine("\nConverted string:");
            Console.WriteLine("To Lowercase: " +
input.ToLower());
            Console.WriteLine("To Uppercase: " +
input.ToUpper());
        }
        else
        {
            Console.WriteLine("No input provided.");
        }
    }
}
```

## Result

PS D:\College\6th Semester\.Net Centric Computing\

Enter a string:

Hello, This IS a test INput

Converted string:

To Lowercase: hello, this is a test input

To Uppercase: HELLO, THIS IS A TEST INPUT

## Experiment 2

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

### Code

```
using System;
class ConsoleApp
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter a string:");
        string ? input = Console.ReadLine();
        if (!string.IsNullOrEmpty(input) && input.Length > 1)
        {
            char firstChar = input[0];
            char lastChar = input[input.Length - 1];
            // Create a new string with first and last characters
            interchanged
            string result = lastChar + input.Substring(1,
input.Length - 2) + firstChar;
            Console.WriteLine("\nOriginal string: " + input);
            Console.WriteLine("New string with first and last
characters interchanged: " + result);
        }
        else
        {
            Console.WriteLine("Please enter a vallid string with
at least two characters.");
        }
    }
}
```

```
}  
}
```

## Result

```
PS D:\College\6th Semester\.Net Centric Computing\Lab Assignmer  
Enter a string:  
Eager
```

```
Original string: Eager  
New string with first and last characters interchanged: rageE
```

## Experiment 3

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

### Code

```
using System;
namespace ConsoleApp
{
    class Vehicle
    {
        public string Brand { get; set; }
        public string Model { get; set; }
        public int Year { get; set; }
        public Vehicle(string brand, string model, int year)
        {
            Brand = brand;
            Model = model;
            Year = year;
        }
        public void DisplayDetails()
        {
            Console.WriteLine($"Brand: {Brand}");
            Console.WriteLine($"Model: {Model}");
            Console.WriteLine($"Year: {Year}");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
```

```

        Console.WriteLine("First Vehicle Object Details:");
        Vehicle car = new Vehicle("Toyota", "Corolla", 2018);
        car.DisplayDetails();
        Console.WriteLine("\nSecond Vehicle Object
Details:");
        Vehicle bike = new Vehicle("Honda", "CBR", 2019);
        bike.DisplayDetails();
    }
}
}

```

## Result

```

PS D:\College\6th Semester\.Net Centric Computir
First Vehicle Object Details:
Brand: Toyota
Model: Corolla
Year: 2018

Second Vehicle Object Details:
Brand: Honda
Model: CBR
Year: 2019

```

## Experiment 4

Write a program to illustrate encapsulation with properties and indexers.

### Code

```
using System;
namespace ConsoleApp
{
    internal class Library
    {
        private string[] books = new string[5];

        public string this[int index]
        {
            get { return books[index]; }
            set { books[index] = value; }
        }

        public int TotalBooks
        {
            get { return books.Length; }
        }

        public string LibraryName { get; set; }

        static void Main(string[] args)
        {
            Library library = new Library { LibraryName = "City
Library" };
            library[0] = "1984 by George Orwell";
        }
    }
}
```



```

        library[1] = "To Kill a Mockingbird by Harper Lee";
        library[2] = "The Great Gatsby by F. Scott
Fitzgerald";
        library[3] = "Art of War by Sun Tzu";
        library[4] = "War and Peace by Leo Tolstoy";

        Console.WriteLine($"{library.LibraryName} Books:");
        for (int i = 0; i < library.TotalBooks; i++)
        {
            Console.WriteLine($"Book {i + 1}: {library[i]}");
        }
    }
}

```

## Result

```

PS D:\College\6th Semester\.Net Centric Computing\Lab Assignment\Lab\c
City Library Books:
Book 1: 1984 by George Orwell
Book 2: To Kill a Mockingbird by Harper Lee
Book 3: The Great Gatsby by F. Scott Fitzgerald
Book 4: Art of War by Sun Tzu
Book 5: War and Peace by Leo Tolstoy

```

## Experiment 5

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

### Code

```
using System;
// Base class
class Animal
{
    public Animal()
    {
        Console.WriteLine("Animal default constructor called.");
    }
    public Animal(string name)
    {
        Console.WriteLine("Animal constructor with parameter
called. Name: " + name);
    }
    public virtual void Speak()
    {
        Console.WriteLine("Animal speaks.");
    }
}

// Derived class
class Dog : Animal
{
    public Dog() : base()
    {
```

```

        Console.WriteLine("Dog default constructor called.");
    }
    public Dog(string name) : base(name)
    {
        Console.WriteLine("Dog constructor with parameter called.
Name: " + name);
    }
    public override void Speak()
    {
        Console.WriteLine("Dog barks.");
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Constructor Overloading
        Console.WriteLine("Constructor Overloading:");
        Animal animal1 = new Animal();
        Animal animal2 = new Animal("Lion");
        Dog dog1 = new Dog();
        Dog dog2 = new Dog("Buddy");

        // Method Overriding
        Console.WriteLine("\nMethod Overriding:");
        Animal animalRef = new Animal();
        animalRef.Speak();
        Animal dogRef = new Dog();
        dogRef.Speak();
    }
}

```

}

## Result

PS D:\College\6th Semester\.Net Centric Computing\Lab Assignment\

Constructor Overloading:

Animal default constructor called.

Animal constructor with parameter called. Name: Lion

Animal default constructor called.

Dog default constructor called.

Animal constructor with parameter called. Name: Buddy

Dog constructor with parameter called. Name: Buddy

Method Overriding:

Animal default constructor called.

Animal speaks.

Animal default constructor called.

Dog default constructor called.

Dog barks.

## Experiment 6

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

### Code

```
using System;

// Define the first interface
public interface IVehicle
{
    void Drive();
}

// Define the second interface
public interface IFlyable
{
    void Fly();
}

// Implement the interfaces in a class
public class Car : IVehicle, IFlyable
{
    public void Drive()
    {
        Console.WriteLine("Car is driving.");
    }

    public void Fly()
    {
        Console.WriteLine("Car is flying.");
    }
}
```

```

    }
}

class Program
{
    static void Main()
    {
        // Create an instance of Car
        Car car = new Car();
        // Call methods from interfaces
        car.Drive();
        car.Fly();
    }
}

```

## Result

```

PS D:\College\6th Semester\.Net Centric Computing\Lab Assignment\
Car is driving.
Car is flying.

```

## Experiment 7

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

### Code

```
using System;
class Program
{
    static void Main()
    {
        try
        {
            // Example: ArgumentNullException
            string str = null;
            PrintString(str); // This line will throw an
exception
        }
        catch (ArgumentNullException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
            // Handle the exception (e.g., provide a default
value)
            Console.WriteLine("String default value: Empty
string");
        }
        catch (Exception ex)
        {
            // Catch-all block for any other exceptions
            Console.WriteLine($"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...");
    }

    static void PrintString(string str)
    {
        if (str == null)
        {
            throw new ArgumentNullException(nameof(str), "String
cannot be null.");
        }
        Console.WriteLine(str);
    }
}

```

## Result

```

PS D:\College\6th Semester\.Net Centric Computing\Lab Assignment\La
Error: String cannot be null. (Parameter 'str')
String default value: Empty string
Program execution completed.
Rest of the program continues...

```



## Experiment 8

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

### Code

```
using System;

internal class DelegateExample
{
    // Step 1: Define a delegate
    public delegate void EventHandler(string message);

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

        // Step 4: Method to raise the event
        public void DoSomething()
        {
            Console.WriteLine("Doing something...");
            OnNotify("Something was done.");
        }

        // Step 5: Method to invoke the event
        protected virtual void OnNotify(string message)
        {
            Notify?.Invoke(message); // Invoke the event
        }
    }
}
```

```

}

// Step 6: Define a class that subscribes to the event
class Subscriber
{
    // Step 7: Method to subscribe to the event
    public void Subscribe(Publisher publisher)
    {
        publisher.Notify += HandleEvent;
    }

    // Step 8: Method to unsubscribe from the event
    public void Unsubscribe(Publisher publisher)
    {
        publisher.Notify -= HandleEvent;
    }

    // Step 9: Event handler method
    private void HandleEvent(string message)
    {
        Console.WriteLine($"Event handled: {message}");
    }
}

// Step 10: Main program execution
class Program
{
    static void Main(string[] args)
    {
        Publisher publisher = new Publisher();
        Subscriber subscriber = new Subscriber();
    }
}

```

```
// Subscribe to the event
subscriber.Subscribe(publisher);

// Trigger the event
publisher.DoSomething();

// Unsubscribe from the event
subscriber.Unsubscribe(publisher);

// Try to trigger the event again
publisher.DoSomething();
    }
}
}
```

## Result

```
PS D:\College\6th Semester\.Net Centric Computing\Lab Assignm
Doing something...
Event handled: Something was done.
Doing something...
```

## Experiment 9

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

### Code

```
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's full. No more items can be
added.");
        }
    }
}
```

```

    }
    // Method to display all items in the list
    public void DisplayItems()
    {
        Console.WriteLine("Listed Items are:");
        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(11);
        intList.Add(12);
        intList.DisplayItems();
        // Creating a list of strings
        GenericList<string> stringList = new
GenericList<string>(3);
        stringList.Add("China");
        stringList.Add("India");
        stringList.DisplayItems();
    }
}

```

## Result

PS D:\College\6th Semester\.Net Centric Computing

Listed Items are:

10

11

12

0

0

Listed Items are:

China

India

## Experiment 10

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

### Code

```
using System;
using System.Collections.Generic;
using System.Linq;
class Program
{
    static void Main()
    {
        List<int> numbers = new List<int> {9,8,7,6,5,4,3,2,1};
        // Using LINQ to filter odd numbers
        IEnumerable<int> oddNumbers = numbers.Where(IsOdd);
        // Display the filtered numbers
        Console.WriteLine("The odd numbers are:");
        foreach (var number in oddNumbers)
        {
            Console.WriteLine(number);
        }
    }
    // Method to check if a number is odd
    static bool IsOdd(int number)
    {
        return number % 2 != 0;
    }
}
```

## Result

PS D:\College\6th Semester\Ne

The odd numbers are:

9

7

5

3

1



## Experiment 11

Write a program to demonstrate Asynchronous programming with async, await, Task in C#.

### Code

```
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($"Async operation completed:
{result}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error occurred: {ex.Message}");
        }

        // Call a synchronous method
        SimulateSyncOperation();
    }
}
```

```

    }

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

    // Synchronous method to simulate work
    static void SimulateSyncOperation()
    {
        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine("Executing synchronous operation");
            // Simulate some work
            Task.Delay(500).Wait(); // Simulate a delay of 0.5
seconds (500 milliseconds)
        }
    }
}

```

## Result

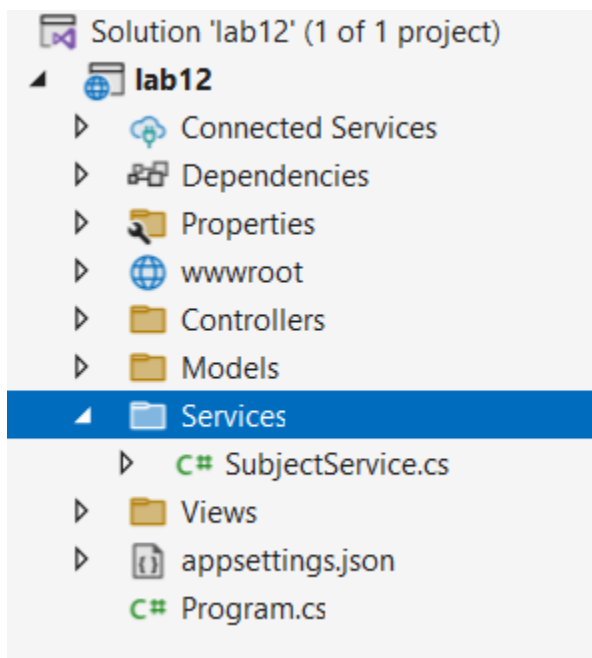
```
PS D:\College\6th Semester\.Net Centric Computing\Lab Assignment\La
Starting asynchronous operation.....
Async operation completed: Operation successfully completed
Executing synchronous operation
Executing synchronous operation
Executing synchronous operation
Executing synchronous operation
Executing synchronous operation
```

## Experiment 12

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

### Steps

1. Firstly, start a new ASP.NET core MVC project.
2. Then, create a Services folder in the workspace, and add new service. In this example, a new service titled SubjectService was added.



3. In the file, SubjectService.cs, create an interface for the service and its basic implementation.

```
namespace lab12.Services
{
    public interface ISubjectService
    {
        string GetRoutine();
    }
}
```

```

    public class SubjectService: ISubjectService
    {
        public string GetRoutine()
        {
            return "The classes today are: CDC, .NET and E-Commerce.";
        }
    }
}

```

4. Configure the dependency injection in the Program.cs file, by adding a scoped service before the app builder as,

```

using lab12.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddScoped<ISubjectService,
SubjectService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}

```

```

        // The default HSTS value is 30 days. You may want to
        change this for production scenarios, see
        https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }

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

    app.UseRouting();

    app.UseAuthorization();

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

    app.Run();

```

The underlined portion showing the added line.

5. Then, the dependency injection needs to be used in a controller. Here, the HomeController.cs is modified to include the new service as dependency.

```

using lab12.Models;
using lab12.Services;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace lab12.Controllers
{

```

```

public class HomeController : Controller
{
    private readonly ISubjectService _services;

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

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

```

6. Edit the Index.cshtml, the view file to reflect the changes as,

```

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

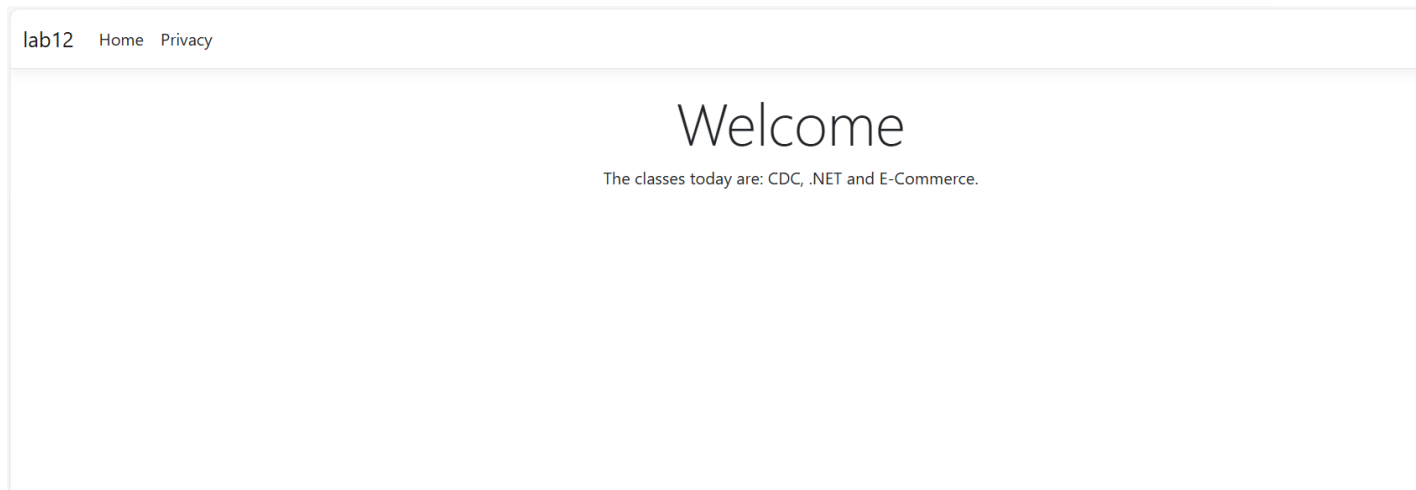
@model string

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>@Model</p>
</div>

```

## Output

Running the project after all the aforementioned changes, we get the following page in the browser.



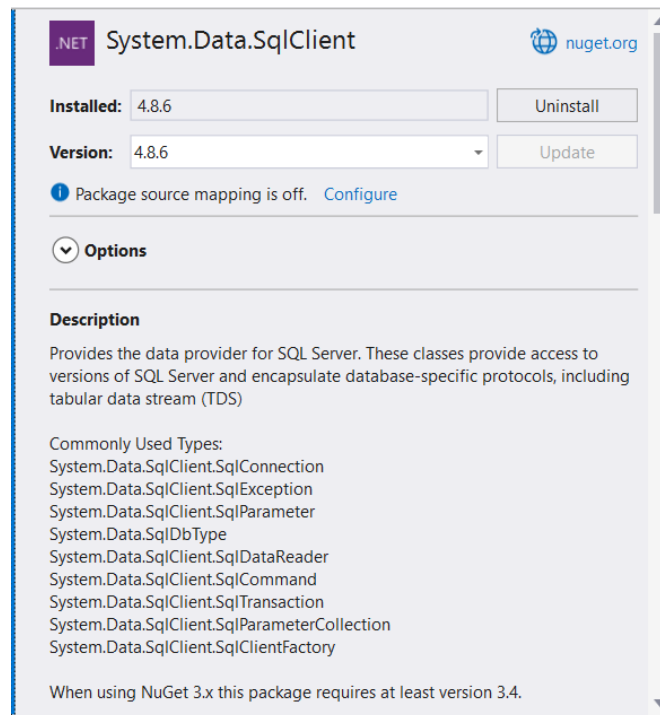


## Experiment 13

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

### Steps

1. First, create a new ASP.NET core MVC application.
2. Since, we are working with SQL Server, we need to install a NuGet package. The exact NuGet package is System.Data.SqlClient, which can be done by right clicking the project in the solution explorer, and clicking manage NuGet packages. Then, we can add the package by browsing through the packages available.



3. Then, create a model to interact with the data in the database. The model is named Items.cs, which contains

```
namespace lab13.Models
{
    public class Item
    {
        public int Id { get; set; }
    }
}
```

```

        public string ? Name { get; set; }
        public string ? Description { get; set; }
        public decimal Quantity { get; set; }

    }
}

```

4. Firstly, we need to setup our SSMS (SQL Server Management Studio) for connection with the database. After running the instance of server from the SSMS, we go the Visual Studio and edit the window view to show SQL Server Object Explorer.

Then, we establish a new connection with the running server and add a new database for our project. For this exercise, a new database named lab\_assignment was added.

To create the table on the new database containing the fields as defined in the model, a right click is done on the database and New Query option is selected, where running the following code creates the needed table.

```

CREATE TABLE Items (
    Id INT PRIMARY KEY IDENTITY,
    Name NVARCHAR(100) NOT NULL,
    Description NVARCHAR(255),
    Quantity DECIMAL(18, 2) NOT NULL
);

```

5. Then, create a new controller named, ItemsController.cs in the Controller folder, that contains the following code,

```

using lab13.Models;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Data.SqlClient;

namespace lab13.Controllers
{

```

```

public class ItemController : Controller
{
    private string _connectionString =
"Server=ARNAV;Database=lab_assignment;Integrated
Security=true;Encrypt=false";

    public ActionResult Index()
    {
        List<Item> items = new List<Item>();
        using (SqlConnection connection = new
SqlConnection(_connectionString))
        {
            connection.Open();
            string query = "SELECT * FROM Items";
            using SqlCommand command = new
SqlCommand(query, connection);
            using SqlDataReader reader =
command.ExecuteReader();
            while (reader.Read())
            {
                Item item = new Item
                {
                    Id = Convert.ToInt32(reader["Id"]),
                    Name = reader["Name"].ToString(),
                    Description =
reader["Description"].ToString(),
                    Quantity =
Convert.ToDecimal(reader["Quantity"])
                };
                items.Add(item);
            }
        }
    }
}

```

```

        }
        return View(items);
    }

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

    [HttpPost]
    public IActionResult Create(Item item)
    {
        using (SqlConnection connection = new
SqlConnection(_connectionString))
        {
            connection.Open();
            string query = "INSERT INTO Items (Name,
Description, Quantity) VALUES (@Name, @Description,
@Quantity)";
            using SqlCommand command = new
SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Name",
item.Name);

            command.Parameters.AddWithValue("@Description",
item.Description);
            command.Parameters.AddWithValue("@Quantity",
item.Quantity);
            command.ExecuteNonQuery();
        }
        return RedirectToAction("Index");
    }

```

```

    }

    public IActionResult Edit(int id)
    {
        Item item = new Item();
        using (SqlConnection connection = new
SqlConnection(_connectionString))
        {
            connection.Open();
            string query = "SELECT * FROM Items WHERE Id
= @Id";

            using SqlCommand command = new
SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Id", id);
            using SqlDataReader reader =
command.ExecuteReader();
            while (reader.Read())
            {
                item.Id = Convert.ToInt32(reader["Id"]);
                item.Name = reader["Name"].ToString();
                item.Description =
reader["Description"].ToString();
                item.Quantity =
Convert.ToDecimal(reader["Quantity"]);
            }
        }
        return View(item);
    }

    [HttpPost]
    public IActionResult Edit(Item item)

```

```

        {
            using (SqlConnection connection = new
SqlConnection(_connectionString))
            {
                connection.Open();
                string query = "UPDATE Items SET Name =
@Name, Description = @Description, Quantity = @Quantity
WHERE Id = @Id";
                using SqlCommand command = new
SqlCommand(query, connection);
                command.Parameters.AddWithValue("@Id",
item.Id);
                command.Parameters.AddWithValue("@Name",
item.Name);

                command.Parameters.AddWithValue("@Description",
item.Description);
                command.Parameters.AddWithValue("@Quantity",
item.Quantity);
                command.ExecuteNonQuery();
            }
            return RedirectToAction("Index");
        }

        public IActionResult Delete(int id)
        {
            using (SqlConnection connection = new
SqlConnection(_connectionString))
            {
                connection.Open();

```

```

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

        using SqlCommand command = new
SqlCommand(query, connection);
        command.Parameters.AddWithValue("@Id", id);
        command.ExecuteNonQuery();
    }
    return RedirectToAction("Index");
}
}
}

```

6. After defining the logics in the controller, respective view pages must also be ready.

#### Index.cshtml

```

@model IEnumerable<lab13.Models.Item>

<h2>Item 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>Description</th>
            <th>Quantity</th>
            <th></th>
        </tr>
    </thead>
</table>

```

```

        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.Id</td>
                <td>@item.Name</td>
                <td>@item.Description</td>
                <td>@item.Quantity</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>

```

### Create.cshtml

```

@model lab13.Models.Item

<h2>Create Item</h2>

<form asp-action="Create">
    <div class="form-group">
        <label asp-for="Name"></label>
        <input asp-for="Name" class="form-control" />
    </div>

```



```

</div>
<div class="form-group">
    <label asp-for="Description"></label>
    <input asp-for="Description" class="form-control" />
</div>
<div class="form-group">
    <label asp-for="Quantity"></label>
    <input asp-for="Quantity" class="form-control" />
</div>
<button type="submit" class="btn btn-
primary">Create</button>
</form>

```

#### Edit.cshtml

```
@model lab13.Models.Item
```

```
<h2>Edit Item</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="Description"></label>
        <input asp-for="Description" class="form-control" />
    </div>
    <div class="form-group">
        <label asp-for="Quantity"></label>

```

```

        <input asp-for="Quantity" class="form-control" />
    </div>
    <button type="submit" class="btn btn-primary">Save
Changes</button>
</form>

```

7. Then, we need to modify the Program.cs file to allow the routing to our new controller, we just need to add the following lines of code

```

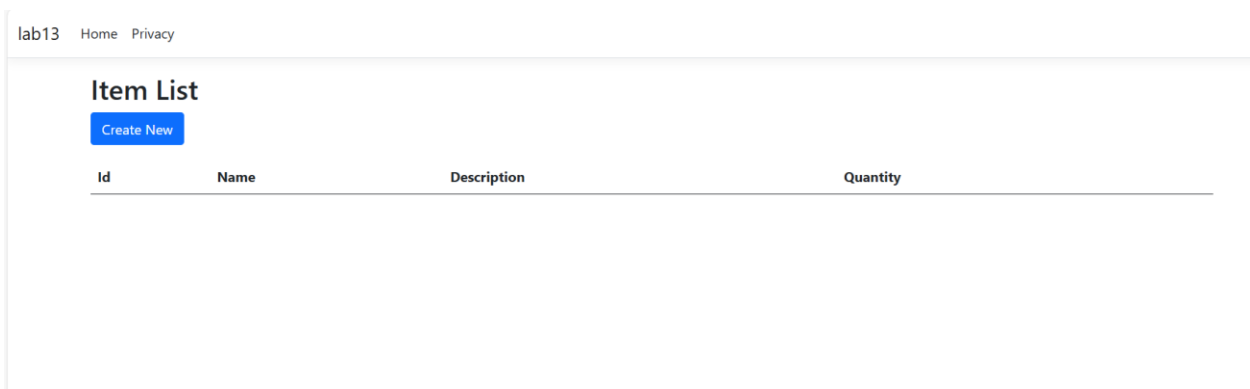
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Item}/{action=Index}/{id?}");
before
app.Run();

```

8. Then, run the project.

## Output

On running the project we get,



lab13 Home Privacy

### Create Item

Name  
wood

Description  
this is wood

Quantity  
32

Create

lab13 Home Privacy

### Edit Item

Name  
apple

Description  
this is an apples

Quantity  
12.00

Save Changes

## Item List

Create New

Id	Name	Description	Quantity	
3	apple	this is an apples	12.00	<a href="#">Edit</a>   <a href="#">Delete</a>
5	wood	this is wood	32.00	<a href="#">Edit</a>   <a href="#">Delete</a>

## Item List

Create New

Id	Name	Description	Quantity	
3	apple	this is an apples	12.00	<a href="#">Edit</a>   <a href="#">Delete</a>

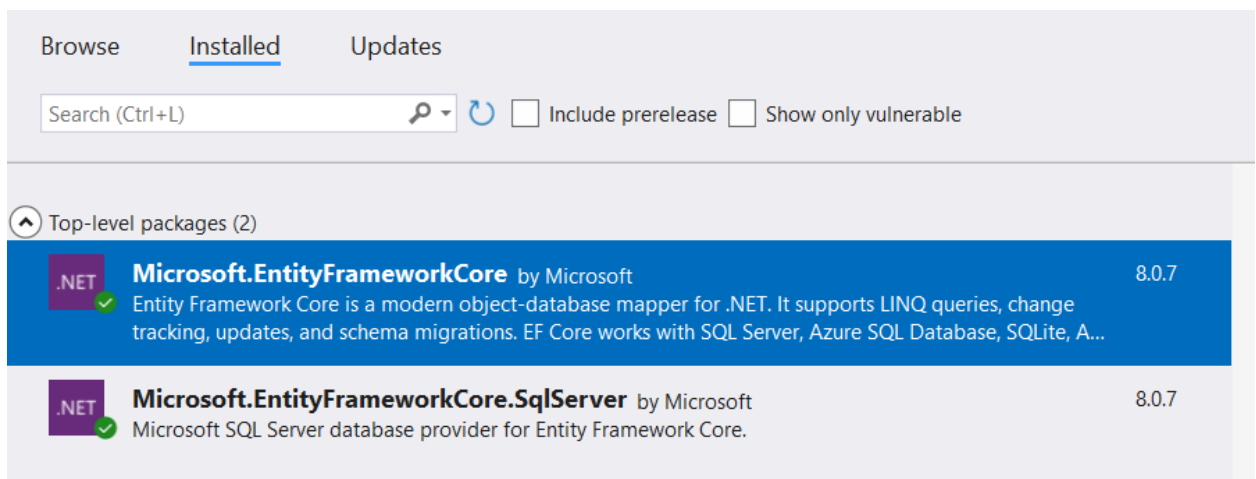
Here, we can conduct the entire CRUD (Create, Read, Update and Delete) operations on the data using ADO.NET.

## Experiment 14

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

### Steps

1. Firstly, start a new ASP.NET core MVC project.
2. Then, add the needed NuGet packages for the project. They are: Microsoft.EntityFrameworkCore and Microsoft.EntityFrameworkCore.SqlServer.



3. Then, add the connection string for connection to the SQL server in the appsettings.json file as,

```
{
  "ConnectionStrings": {
    "SQLServerConnection":
    "Server=ARNAV;Database=lab_assignment;Integrated
    Security=True;TrustServerCertificate=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

```

    }
},
"AllowedHosts": "*"
}

```

4. After setting the connection string, connect to the server instance from the SQL Server Object Explorer, and run a query to create the table in the database as,

```

CREATE TABLE Employees (
    Id INT PRIMARY KEY IDENTITY(1,1),
    Name NVARCHAR(100),
    Age DECIMAL(3, 0),
    Salary DECIMAL(18, 2)
);

```

5. Then, define the employee model as per the table,

```

using System.ComponentModel.DataAnnotations;

namespace lab14.Models
{
    public class Employee
    {
        [Key]
        public int Id { get; set; }
        public string? Name { get; set; }
        public decimal Age { get; set; }
        public decimal Salary { get; set; }
    }
}

```

6. Define the controller with all the needed methods as follows:

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

```

```

using System.Threading.Tasks;
using lab14.Data;
using lab14.Models;

namespace EmployeeApp.Controllers
{
    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,Age,Salary")] Employee employee)

```

```

        {
            if (ModelState.IsValid)
            {
                _context.Add(employee);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(employee); // Pass the model back to
the view if it's invalid
        }
    }
}

```

7. Define the views as follows,

Index.cshtml

```

@model IEnumerable<lab14.Models.Employee>
<!DOCTYPE html>
<html>
<head>
    <title>Employee List</title>
    <style>
        body { font-family: Arial, sans-serif; background-
color: #f4f4f4; margin: 0; padding: 0; }
        .container { max-width: 800px; margin: 20px auto;
padding: 20px; background: #fff; border-radius: 8px; box-
shadow: 0 0 10px rgba(0, 0, 0, 0.1); }
        h2 { color: #333; text-align: center; margin-bottom:
20px; }
    </style>

```

```

        table { width: 100%; border-collapse: collapse;
margin-bottom: 20px; }
        table, th, td { border: 1px solid #ddd; }
        th, td { padding: 10px; text-align: left; }
        th { background-color: #007bff; color: #fff; }
        tr:nth-child(even) { background-color: #f2f2f2; }
        .btn-primary { background-color: #007bff; color:
#fff; border: none; padding: 10px 15px; border-radius: 4px;
text-decoration: none; display: inline-block; }
        .btn-primary:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="container">
        <h2>Employee List</h2>
        <table>
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Age</th>
                    <th>Salary</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td>@item.Name</td>
                        <td>@item.Age</td>
                        <td>@item.Salary</td>
                    </tr>
                }
            </tbody>
        </table>
    </div>

```



```

        }
    </tbody>
</table>
    <a href="@Url.Action("Create")" class="btn btn-
primary">Create New</a>
</div>
</body>
</html>

```

#### Create.cshtml

```

@model lab14.Models.Employee
<!DOCTYPE html>
<html>
<head>
    <title>Create Employee</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 0;
        }

        .container {
            max-width: 600px;
            margin: 0 auto;

```

```

padding: 20px;
background: #fff;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h2 {
color: #333;
text-align: center;
}

.form-group {
margin-bottom: 15px;
}

.form-group label {
display: block;
font-weight: bold;
margin-bottom: 5px;
}

.form-group input {
width: 100%;
padding: 10px;
border: 1px solid #ddd;
border-radius: 4px;
box-sizing: border-box;
}

.form-group input:focus {
border-color: #007bff;

```

```

        outline: none;
    }

    .form-group .text-danger {
        color: #dc3545;
    }

    .btn-primary {
        background-color: #007bff;
        color: #fff;
        border: none;
        padding: 10px 15px;
        border-radius: 4px;
        cursor: pointer;
    }

    .btn-primary:hover {
        background-color: #0056b3;
    }

    .btn-secondary {
        background-color: #6c757d;
        color: #fff;
        border: none;
        padding: 10px 15px;
        border-radius: 4px;
        cursor: pointer;
    }

    .btn-secondary:hover {
        background-color: #5a6268;
    }

```

```

    }
    </style>
</head>
<body>
    <div class="container">
        <h2>Add 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="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">
                <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="Add" class="btn
btn-primary" />
    </div>
</form>
    <a asp-action="Index" class="btn btn-secondary">Back
to List</a>
</div>
</body>
</html>

```

8. Add a configuration for DbContext in the Program.cs file as,

```

// Configure DbContext with SQL Server
builder.Services.AddDbContext<ApplicationDbContext>(options
=>
{

    options.UseSqlServer(builder.Configuration.GetConnectionStri
ng("SQLServerConnection"));
    options.EnableSensitiveDataLogging();
});

```

9. Run the project and observe in the browser.

## Output

Running the project after all the aforementioned changes, we get the following page in the browser.

### Add Employee

Name

Ram

Age

23

Salary

13242

Add

Back to List

### Employee List

Name	Age	Salary
as	12	321.00
Ram	23	13242.00

Create New

## Experiment 15

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

### Steps

1. Firstly, start a new ASP.NET core MVC project.
2. Create a StateController to handle all the necessary actions as,

```
using Microsoft.AspNetCore.Mvc;

namespace lab15.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();
    }
}
}

```

3. Add two new views for the actions,

#### Add.cshtml

```

@model lab15.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>

```

4. Modify the Program.cs to allow session by adding the following underlined portions,

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
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();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to
    change this for production scenarios, see
    https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

```

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

app.UseRouting();

app.UseAuthorization();

app.UseSession();

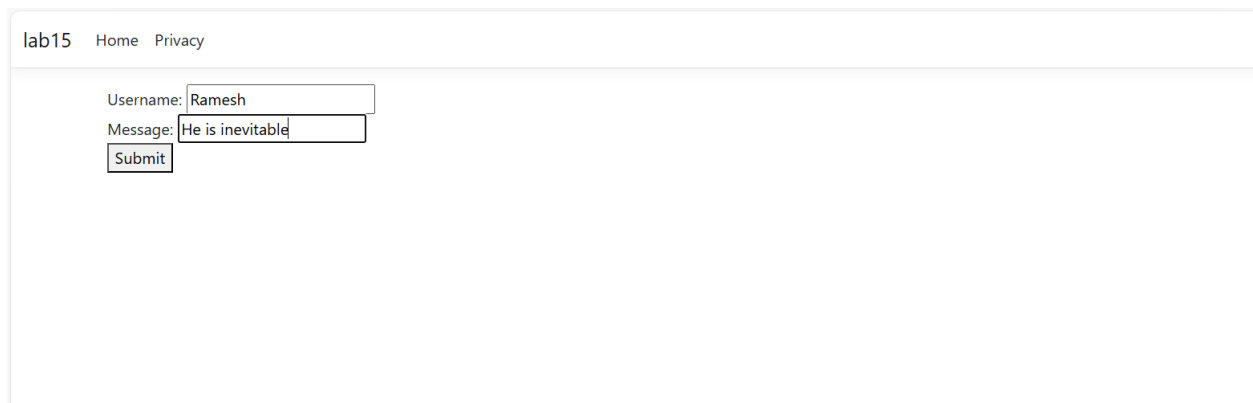
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=State}/{action=Index}/{id?}");

app.Run();
```

5. Then, run the project in browser and observe the effects.

## Output

Running the project after all the aforementioned changes, we get the following page in the browser.



The screenshot shows a web browser window with a light gray header bar containing the text "lab15 Home Privacy". Below the header, there is a form with two input fields and a submit button. The first field is labeled "Username:" and contains the text "Ramesh". The second field is labeled "Message:" and contains the text "He is inevitable". Below these fields is a button labeled "Submit".

## Display

Username from Session State: Ramesh

Message from TempData: He is inevitable

## Experiment 16

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

### Steps

1. Firstly, start a new ASP.NET core MVC project.
2. Then, setup a controller named StateController for all the needed logics as,

```
using Microsoft.AspNetCore.Mvc;

namespace lab16.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); //
Cookie expiration time
            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();
        }
    }
}

```

3. Add the razor files as,

Index.cshtml

```

@page
@model lab16.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>

```

GetCookie.cshtml

```

@page
@model lab16.Controllers.StateController
<h2>Stored User Data:</h2>
<p>@ViewBag.UserData</p>

```

## Output

Running the project after all the aforementioned changes, we get the following page in the browser.

