15. Create a web form that contains two label that display Enter first number and enter second number, two text box for taking an input, third text box for output and three button add, subtract and find prime. Add proper validation like text box should not be empty, value of first field should be greater than value of second field. If add button is clicked display the addition of two number given in textboxes, if subtract button is clicked display the subtraction of two number given in textboxes and if findprime is clicked then display the prime number from first value to second value given in textboxes.

Code:

Lab15.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Lab15.aspx.cs" Inherits="ncc.Properties.Lab" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

   <title></title>

</head>

<body>

    <form id="form1" runat="server">

      <div class="container">

          <h1>Number Operations</h1>

          <label for="firstNumber">Enter first number:</label>

          <asp:TextBox ID="firstNumber" runat="server"></asp:TextBox>

          <br />

          <label for="secondNumber">Enter second number:</label>

          <asp:TextBox ID="secondNumber" runat="server"></asp:TextBox>

          <br />

          <asp:Button ID="btnAdd" runat="server" Text="Add" OnClick="btnAdd_Click" />

          <asp:Button ID="btnSubtract" runat="server" Text="Subtract" OnClick="btnSubtract_Click" />

          <asp:Button ID="btnFindPrime" runat="server" Text="Find Prime" OnClick="btnFindPrime_Click" />
```

```
            <br />
            <label for="output">Output:</label>
            <asp:TextBox ID="output" runat="server" ReadOnly="true"></asp:TextBox>
        </div>
    </form>
</body>
</html>
```

Lab15.aspx.cs

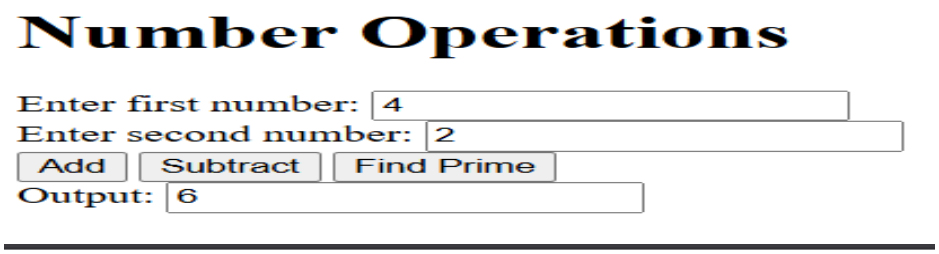
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace ncc.Properties
{
    public partial class Lab : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        private bool ValidateInputs(out int firstNum, out int secondNum, string operation)
        {
            firstNum = 0;
            secondNum = 0;
            if (string.IsNullOrWhiteSpace(firstNumber.Text) ||
string.IsNullOrWhiteSpace(secondNumber.Text))
            {
```

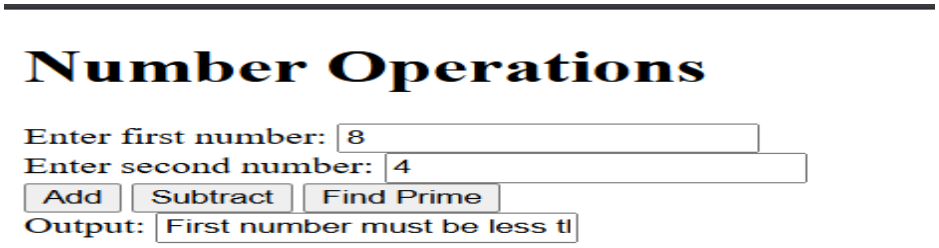
```
            output.Text = "Both fields are required.";
            return false;
        }
        if (!int.TryParse(firstNumber.Text, out firstNum) || !int.TryParse(secondNumber.Text,
out secondNum))
        {
            output.Text = "Please enter valid numbers.";
            return false;
        }
        if (operation == "Add" || operation == "Subtract")
        {
            if (firstNum <= secondNum)
            {
                output.Text = "First number must be greater than second number.";
                return false;
            }
        }
        else if (operation == "FindPrime")
        {
            if (firstNum > secondNum)
            {
                output.Text = "First number must be less than or equal to second number.";
                return false;
            }
        }
        return true;
    }
    protected void btnAdd_Click(object sender, EventArgs e)
    {
```

```
    if (ValidateInputs(out int firstNum, out int secondNum, "Add"))
    {
        int result = firstNum + secondNum;
        output.Text = result.ToString();
    }
}
protected void btnSubtract_Click(object sender, EventArgs e)
{
    if (ValidateInputs(out int firstNum, out int secondNum, "Subtract"))
    {
        int result = firstNum - secondNum;
        output.Text = result.ToString();
    }
}
protected void btnFindPrime_Click(object sender, EventArgs e)
{
    if (ValidateInputs(out int firstNum, out int secondNum, "FindPrime"))
    {
        List<int> primes = FindPrimesInRange(firstNum, secondNum);
        output.Text = string.Join(", ", primes);
    }
}
private List<int> FindPrimesInRange(int start, int end)
{
    List<int> primes = new List<int>();
    for (int num = start; num <= end; num++)
    {
        if (IsPrime(num))
```

```
            {
                primes.Add(num);
            }
        }
        return primes;
    }
    private bool IsPrime(int number)
    {
        if (number <= 1) return false; // 0 and 1 are not prime numbers
        if (number == 2) return true; // 2 is the only even prime number
        if (number % 2 == 0) return false; // other even numbers are not prime
        for (int i = 3; i <= Math.Sqrt(number); i += 2)
        {
            if (number % i == 0)
            {
                return false; // found a divisor, not prime
            } }
        return true; // no divisors found, it's prime
    }
} }
```

Output:





16. Write a console program (ADO.net) to create a table tbl_registration that have fields (id int primary key, username, password, repassword, gender, course and country). After this perform the following operation

- Insert any 5 data into tbl_registration. All the required input should be taken from user
- Display all the record of database table
- Update the name and course of a person to data given by user according to id given by user
- Delete the record of person whose id is given by user
- Display all the record of person who are male and also from country Nepal

CRUDoperation.cs

```
using System;

using System.Data.SqlClient;

namespace ncc1.Properties

{

   internal class CRUDoperation

   {

      string cs = "Data Source=DESKTOP-U7SCV47\\SQLEXPRESS;Initial
Catalog=db_nccsb;Integrated Security=True;TrustServerCertificate=True";

      public void CreateTable()

      {

         try

         {

            using (SqlConnection sc = new SqlConnection(cs))

            {

               sc.Open();

               string createTableQuery = "CREATE TABLE tbl_registration (id INT PRIMARY
               KEY, username VARCHAR(50), password VARCHAR(50), repassword
               VARCHAR(50), gender VARCHAR(50), course VARCHAR(50), country
               VARCHAR(50))";

               SqlCommand cmd = new SqlCommand(createTableQuery, sc);
```

```
            cmd.ExecuteNonQuery();
            Console.WriteLine("Table created successfully!!");
        }
    }
    catch (SqlException ex)
    {
        Console.WriteLine("SQL Error: " + ex.Message);
    }
}
public void InsertRecord()
{
    try
    {
        Console.Write("ID: ");
        int id = int.Parse(Console.ReadLine());
        Console.Write("Username: ");
        string username = Console.ReadLine();
        Console.Write("Password: ");
        string password = Console.ReadLine();
        Console.Write("Re-enter Password: ");
        string repassword = Console.ReadLine();
        Console.Write("Gender: ");
        string gender = Console.ReadLine();
        Console.Write("Course: ");
        string course = Console.ReadLine();
        Console.Write("Country: ");
        string country = Console.ReadLine();
        using (SqlConnection sc = new SqlConnection(cs))
```

```
        {
            sc.Open();
            string insertQuery = "INSERT INTO tbl_registration (id, username, password,
              repassword, gender, course, country) VALUES (@id, @username, @password,
              @repassword, @gender, @course, @country)";
            SqlCommand cmd = new SqlCommand(insertQuery, sc);
            cmd.Parameters.AddWithValue("@id", id);
            cmd.Parameters.AddWithValue("@username", username);
            cmd.Parameters.AddWithValue("@password", password);
            cmd.Parameters.AddWithValue("@repassword", repassword);
            cmd.Parameters.AddWithValue("@gender", gender);
            cmd.Parameters.AddWithValue("@course", course);
            cmd.Parameters.AddWithValue("@country", country);
            cmd.ExecuteNonQuery();
            Console.WriteLine("Record inserted successfully!!");
        }
    }
    catch (SqlException ex)
    {
        Console.WriteLine("SQL Error: " + ex.Message);
    }
}
public void DisplayAllRecords()
{
    try
    {
        using (SqlConnection sc = new SqlConnection(cs))
      {
            sc.Open();
```

```
            string disQuery = "SELECT * FROM tbl_registration";
            SqlCommand cmd = new SqlCommand(disQuery, sc);
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                Console.WriteLine($"ID: {reader["id"]}, Username: {reader["username"]},
Gender: {reader["gender"]}, Course: {reader["course"]}, Country: {reader["country"]}");
            }
        }
    }
    catch (SqlException ex)
    {
        Console.WriteLine("SQL Error: " + ex.Message);
    }
}
public void UpdateRecord()
{
    try
    {
        Console.Write("Enter ID of the user to update: ");
        int updateId = int.Parse(Console.ReadLine());
        Console.Write("Enter new username: ");
        string newUsername = Console.ReadLine();
        Console.Write("Enter new course: ");
        string newCourse = Console.ReadLine();
        using (SqlConnection sc = new SqlConnection(cs))
        {
            sc.Open();
```

```
            string updateQuery = "UPDATE tbl_registration SET username = @username,
            course = @course WHERE id = @id";
            SqlCommand cmd = new SqlCommand(updateQuery, sc);
            cmd.Parameters.AddWithValue("@id", updateId);
            cmd.Parameters.AddWithValue("@username", newUsername);
            cmd.Parameters.AddWithValue("@course", newCourse);
            int rowsAffected = cmd.ExecuteNonQuery();
            if (rowsAffected > 0)
                Console.WriteLine("Record updated successfully!!");
            else
                Console.WriteLine("No record found with the given ID.");
        }
    }
    catch (SqlException ex)
    {
        Console.WriteLine("SQL Error: " + ex.Message);
    }
}
public void DeleteRecord()
{
    try
    {
        Console.Write("Enter ID of the user to delete: ");
        int deleteId = int.Parse(Console.ReadLine());
        using (SqlConnection sc = new SqlConnection(cs))
        {
            sc.Open();
            string deleteQuery = "DELETE FROM tbl_registration WHERE id = @id";
            SqlCommand cmd = new SqlCommand(deleteQuery, sc);
```

```
            cmd.Parameters.AddWithValue("@id", deleteId);
            int rowsAffected = cmd.ExecuteNonQuery();

            if (rowsAffected > 0)
                Console.WriteLine("Record deleted successfully!!");
            else
                Console.WriteLine("No record found with the given ID.");
        }
    }
    catch (SqlException ex)
    {
        Console.WriteLine("SQL Error: " + ex.Message);
    }
}
public void DisplayMaleUsersFromNepal()
{
    try
    {
        using (SqlConnection sc = new SqlConnection(cs))
        {
            sc.Open();
            string selectQuery = "SELECT * FROM tbl_registration WHERE gender = 'Male'
            AND country = 'Nepal'";
            SqlCommand cmd = new SqlCommand(selectQuery, sc);
            SqlDataReader reader = cmd.ExecuteReader();
            Console.WriteLine("Male Users from Nepal:");
            while (reader.Read())
            {
```

```
                Console.WriteLine($"ID: {reader["id"]}, Username: {reader["username"]},
                Course: {reader["course"]}");
            }
        }
        catch (SqlException ex)
        {
            Console.WriteLine("SQL Error: " + ex.Message);
        }
    }
}
```

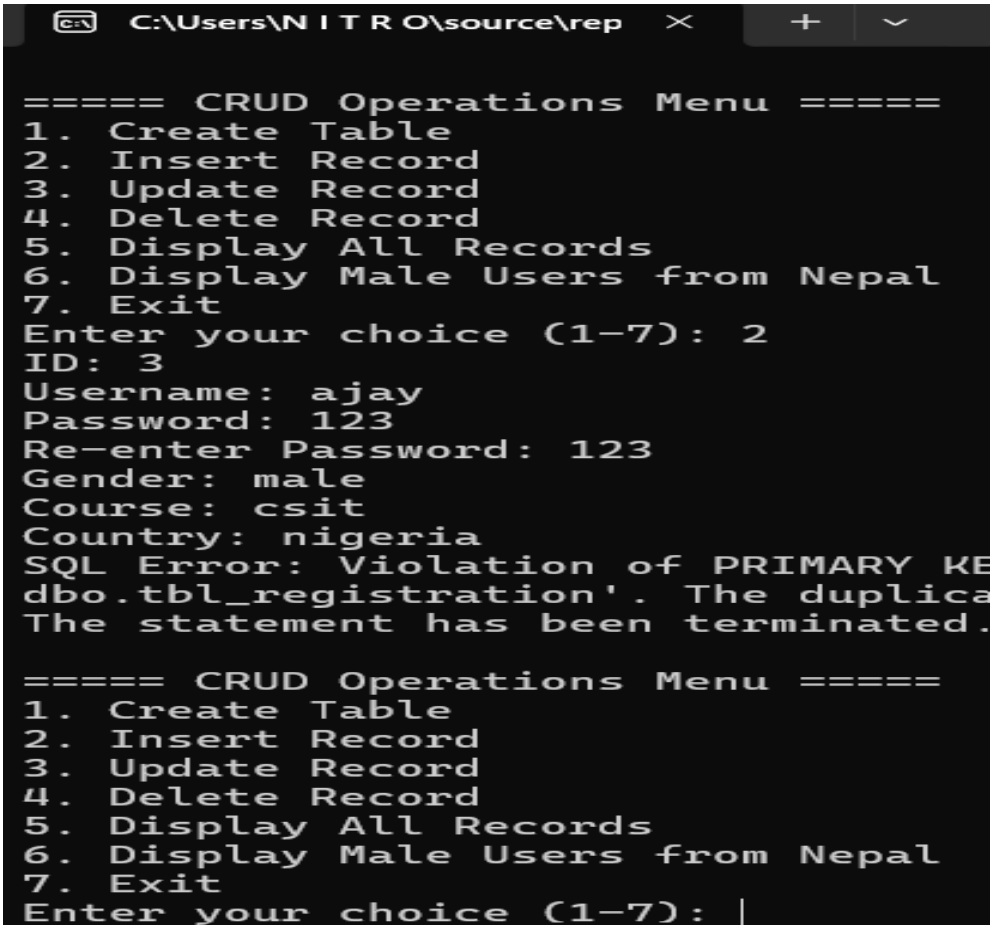
Program.cs

```
using ncc1.Properties;
using System;
namespace ncc1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            CRUDoperation c1 = new CRUDoperation();
            bool exit = false;
            while (!exit)
            {
                Console.WriteLine("\n===== CRUD Operations Menu =====");
                Console.WriteLine("1. Create Table");
                Console.WriteLine("2. Insert Record");
```

```
Console.WriteLine("3. Update Record");
Console.WriteLine("4. Delete Record");
Console.WriteLine("5. Display All Records");
Console.WriteLine("6. Display Male Users from Nepal");
Console.WriteLine("7. Exit");
Console.Write("Enter your choice (1-7): ");

string choice = Console.ReadLine();
switch (choice)
{
    case "1":
        c1.CreateTable();
        break;
    case "2":
        c1.InsertRecord();
        break;
    case "3":
        c1.UpdateRecord();
        break;
    case "4":
        c1.DeleteRecord();
        break;
    case "5":
        c1.DisplayAllRecords();
        break;
    case "6":
        c1.DisplayMaleUsersFromNepal();
        break;
```

```
                case "7":
                    exit = true;
                    Console.WriteLine("Exiting program...");
                    break;
                default:
                    Console.WriteLine("Invalid choice! Please enter a number between 1 and 7.");
                    break;
            }
        }
    }
  }
}
```

OUTPUT:



17. For the table created in question no. 3, create a web form for registration which should contains username, password, repassword, gender (radio button), course (checkbox) and country (dropdown) and submit button. When submit is pressed insert the value given by user into database table. Use proper validation: username, password and repassword should not be empty, item of radio button, checkbox and dropdown menu should be selected.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebLabb.aspx.cs" Inherits="ncc2.Properties.WebLab" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

    <title>User Registration</title>

    <!-- Include jQuery and validation scripts -->

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js" type="text/javascript"></script>

    <script src="https://ajax.aspnetcdn.com/ajax/4.5/1/modernizr-2.8.3.js" type="text/javascript"></script>

    <script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.13.1/jquery.validate.min.js" type="text/javascript"></script>

    <script src="https://ajax.aspnetcdn.com/ajax/jquery.validate.unobtrusive/3.2.5/jquery.validate.unobtrusive.min.js" type="text/javascript"></script>

    <script type="text/javascript">

        // Ensure unobtrusive validation works

        $(document).ready(function () {

            $.validator.setDefaults({

                ignore: ':hidden'

            });

        });

    </script>

</head>

<body>
```

```
<form id="form1" runat="server">
    <h2>User Registration</h2>
    <label>Username:</label>
    <asp:TextBox ID="txtUsername" runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator ControlToValidate="txtUsername" ErrorMessage="Username
    is required!" ForeColor="Red" runat="server" /><br><br>
    <label>Password:</label>
    <asp:TextBox ID="txtPassword" runat="server" TextMode="Password"></asp:TextBox>
    <asp:RequiredFieldValidator ControlToValidate="txtPassword" ErrorMessage="Password
    is required!" ForeColor="Red" runat="server" /><br><br>
    <label>Re-enter Password:</label>
    <asp:TextBox ID="txtRepassword" runat="server"
    TextMode="Password"></asp:TextBox>
    <asp:RequiredFieldValidator ControlToValidate="txtRepassword" ErrorMessage="Re-
    enter password!" ForeColor="Red" runat="server" />
    <asp:CompareValidator ControlToValidate="txtRepassword"
    ControlToCompare="txtPassword" ErrorMessage="Passwords do not match!"
    ForeColor="Red" runat="server" /><br><br>
    <asp:Label ID="l6" Text="Gender" runat="server"></asp:Label>
    <asp:RadioButton ID="rb1" GroupName="gender" Text="Male" runat="server" />
    <asp:RadioButton ID="rb2" GroupName="gender" Text="Female" runat="server" />
    <br />
    <label>Course:</label>
    <asp:CheckBox ID="csit" runat="server" Text="CSIT" />
    <asp:CheckBox ID="bbm" runat="server" Text="BBM" />
    <asp:CheckBox ID="bim" runat="server" Text="BIM" />
    <br />

    <label>Country:</label>
    <asp:DropDownList ID="ddlCountry" runat="server">
```

```
            <asp:ListItem Text="Select Country" Value="" />
            <asp:ListItem Text="Nepal" Value="Nepal" />
            <asp:ListItem Text="India" Value="India" />
            <asp:ListItem Text="USA" Value="USA" />
        </asp:DropDownList>
        <asp:RequiredFieldValidator ControlToValidate="ddlCountry" InitialValue=""
        ErrorMessage="Select a country!" ForeColor="Red" runat="server" /><br><br>
        <asp:Button ID="btnSubmit" runat="server" Text="Register" OnClick="btnSubmit_Click"
/>
    </form>
</body>
</html>
```

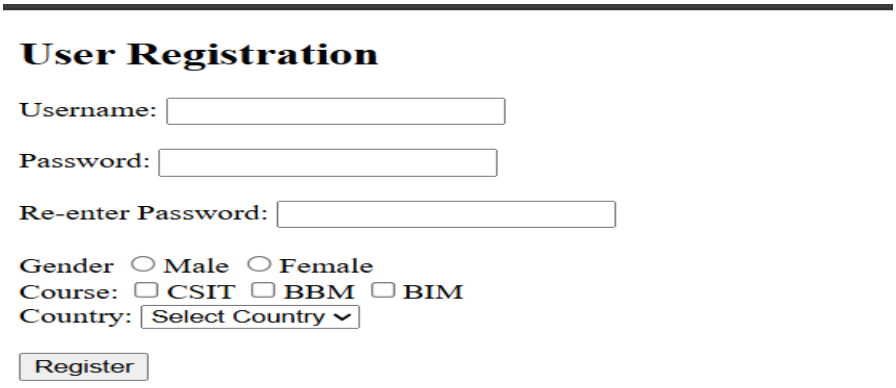
Weblabb.aspx.cs

```
using System;
using System.Data.SqlClient;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace ncc2.Properties
{
    public partial class WebLab : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            string username = txtUsername.Text.Trim();
            string password = txtPassword.Text.Trim();
```

```
string repassword = txtRepassword.Text.Trim();

string gender = rb1.Checked ? "Male" : "Female”;

string courses = "";

if (csit.Checked) courses += "CSIT, ";

if (bbm.Checked) courses += "BBM, ";

if (bim.Checked) courses += "BIM, ";

courses = courses.TrimEnd(',', ' ');

string country = ddlCountry.SelectedValue;

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password) ||
string.IsNullOrEmpty(repassword))

{

   Response.Write("<script>alert('All fields are required!');</script>");

   return;

}

if (password != repassword)

{

   Response.Write("<script>alert('Passwords do not match!');</script>");

   return;

}

string connString = "Data Source=DESKTOP-U7SCV47\\SQLEXPRESS;Initial
Catalog=db_nccsb;Integrated Security=True;TrustServerCertificate=True";

using (SqlConnection conn = new SqlConnection(connString))

{

   conn.Open();

   Random rand = new Random();

   int randomId = rand.Next(10000, 99999);

   string query = "INSERT INTO tbl_registration (id, username, password, repassword,
   gender, course, country) " +

    "VALUES (@id, @username, @password, @repassword, @gender, @course,
  @country)";
```

```
using (SqlCommand cmd = new SqlCommand(query, conn))
{
   cmd.Parameters.AddWithValue("@id", randomId);
   cmd.Parameters.AddWithValue("@username", username);
   cmd.Parameters.AddWithValue("@password", password);
   cmd.Parameters.AddWithValue("@repassword", repassword);
   cmd.Parameters.AddWithValue("@gender", gender);
   cmd.Parameters.AddWithValue("@course", courses);
   cmd.Parameters.AddWithValue("@country", country);
   int rows = cmd.ExecuteNonQuery();
   if (rows > 0)
   {
      Response.Write("<script>alert('Registration Successful!');</script>");
   }
   else
   {
      Response.Write("<script>alert('Error in registration!');</script>");
   } }}}}}
```

Output:



18. Demonstrate model, view and controller by showing different action method, views, model, accessing controller, model and view.

StudentController.cs

```
using Microsoft.AspNetCore.Mvc;
using lab1.Models;
namespace lab1.Controllers
{
   public class StudentController : Controller
   {
      public IActionResult Details()
      {
         Student s1 = new Student
         {
            Id = 24,
            Name = "Student",
            Faculty = "Science"
         };
         return View(s1);
      }
   }
}
```

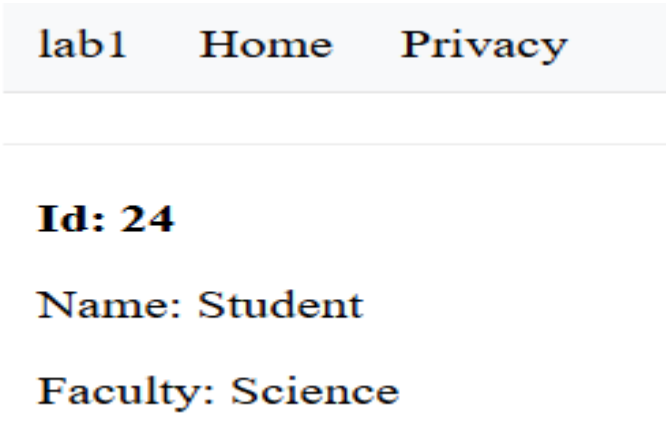
Details.cshtml

```
@ model lab1.Models.Student

<h2>Id: @Model.Id</h2>
<h2>Name: @Model.Name</h2>
<h2>Faculty: @Model.Faculty</h2>
```

lab1.Models.Student

```
namespace lab1.Models
{
   public class Student
   {
      private int id;
      private string name;
      private string faculty;
      public int Id { get { return id; } set { id = value; } }
      public string Name { get { return name; } set { name = value; } }
      public string Faculty { get { return faculty; } set { faculty = value; } }
   }
}
```

OUTPUT:



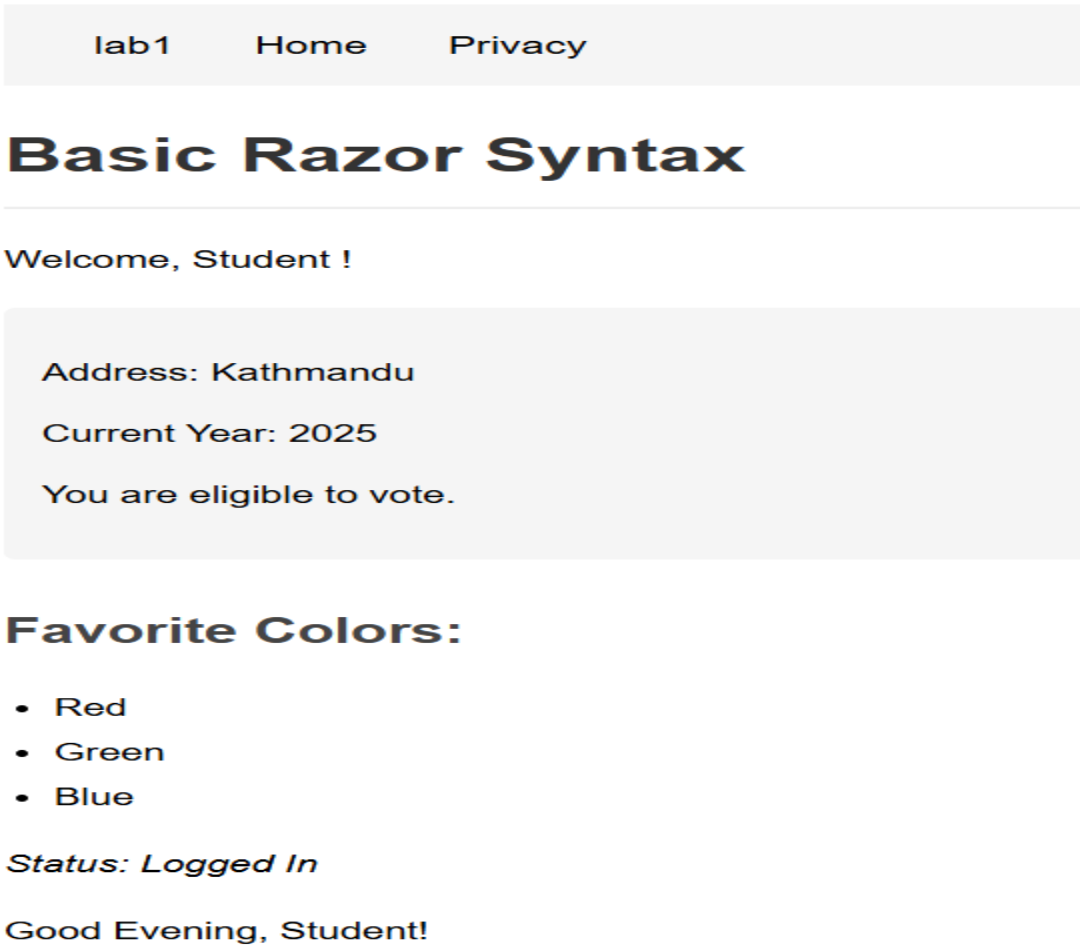
19. Demonstrate use of razor syntax.

Views/Home/Index.cshtml

```
@{
    var name = "Student ";
    int age = 22;
    var address = "tokha,Kathmandu";
    var isLoggedIn = true;
    var colors = new List<string> { "Red", "Green", "Blue" };
}
<h2>Basic Razor Syntax</h2>
<p>Welcome, @name!</p>
<p>Address: @address</p>
<p>Current Year: @DateTime.Now.Year</p>
@if (age >= 18)
{
    <p>You are eligible to vote.</p>
}
else
{
    <p>You are not eligible to vote.</p>
}
<h3>Favorite Colors:</h3>
<ul>
    @foreach (var color in colors)
    {
        <li>@color</li>
    }
```

```
</ul>
<p>Status: @(isLoggedIn ? "Logged In" : "Guest")</p>
@{
    string greeting = DateTime.Now.Hour < 12 ? "Good Morning" : "Good Evening";
}
<p>@greeting, @name!</p>
```

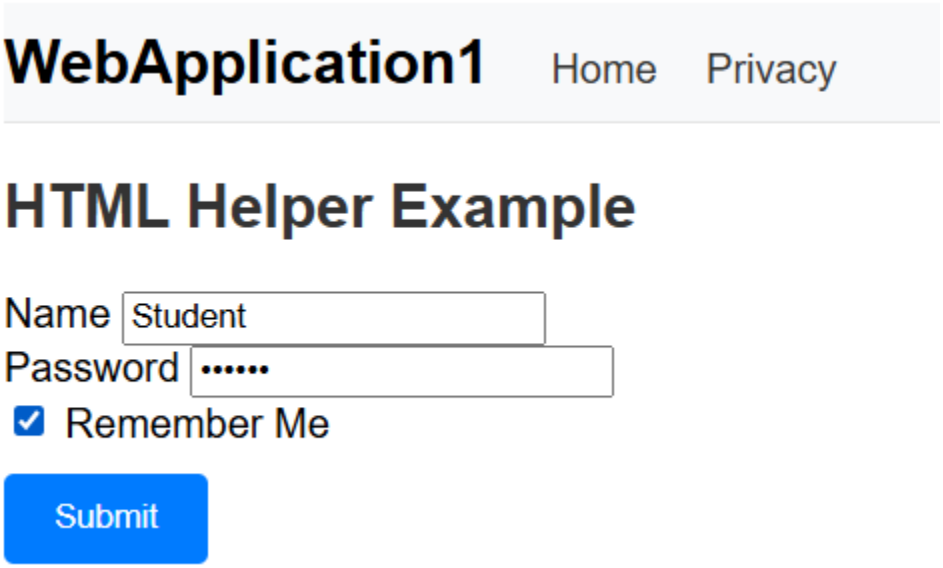
Output:



20. Demonstrate use of html tag helper.

```
@{
    ViewData["Title"] = "HTML Helper Example";
}

<h2>HTML Helper Example</h2>

@using (Html.BeginForm())
{
    @Html.Label("Name")
    @Html.TextBox("Name")

    <br />

    @Html.Label("Password")
    @Html.Password("Password")

    <br />

    @Html.Label("Remember Me")
    @Html.CheckBox("RememberMe")

    <br />

    <input type="submit" value="Submit" />
}
```

Output:


WebApplication1
Home
Privacy
HTML Helper Example
Name
Student
Password
Remember Me
Submit

21. Using Entity framework create a table tbl_officer having field (id, name, gender, phone, department and position) after this perform complete CRUDE operation (insert, update, display and delete). User proper validation.

Officecontroller.cs

```
using Microsoft.AspNetCore.Mvc;
using EFCoreNCCSB.Models;

using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

namespace EFCoreNCCSB.Controllers
{
    public class OfficerController : Controller
    {
        private readonly ApplicationDbContext _context;

        public OfficerController(ApplicationDbContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            var officers = await _context.Officers.ToListAsync();
            return View(officers); // Pass the list to the view
        }
        public IActionResult Create()
        {
            return View();
```

```
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Officer officer)
{
    if (ModelState.IsValid)
    {
        _context.Add(officer);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(officer);
}
public async Task<IActionResult> Edit(int id)
{
    var officer = await _context.Officers.FindAsync(id);
    if (officer == null) return NotFound();
    return View(officer);
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, Officer officer)
{
    if (id != officer.Id) return NotFound();
    if (ModelState.IsValid)
    {
        _context.Update(officer);
        await _context.SaveChangesAsync();
```

```
            return RedirectToAction(nameof(Index));
        }
        return View(officer);
    }
    public async Task<IActionResult> Delete(int id)
    {
        var officer = await _context.Officers.FindAsync(id);
        if (officer == null) return NotFound();
        return View(officer);
    }
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var officer = await _context.Officers.FindAsync(id);
        if (officer != null)
        {
            _context.Officers.Remove(officer);
            await _context.SaveChangesAsync();
        }
        return RedirectToAction(nameof(Index));
    }
  }
}
```

Officer.cs

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

```
namespace EFCoreNCCSB.Models
{
    public class Officer
    {
        [Key]
        public int Id { get; set; } // Primary Key (Auto-increment)

        [Required(ErrorMessage = "Name is required")]
        [StringLength(100, ErrorMessage = "Name cannot exceed 100 characters")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Gender is required")]
        [RegularExpression("^(Male|Female|Other)$", ErrorMessage = "Gender must be Male, Female, or Other")]
        public string Gender { get; set; }
        [Required(ErrorMessage = "Phone number is required")]
        [Phone(ErrorMessage = "Invalid phone number")]
        public string Phone { get; set; }
        [Required(ErrorMessage = "Department is required")]
        [StringLength(50, ErrorMessage = "Department name cannot exceed 50 characters")]
        public string Department { get; set; }
        [Required(ErrorMessage = "Position is required")]
        [StringLength(50, ErrorMessage = "Position name cannot exceed 50 characters")]
        public string Position { get; set; }
    }
}
```

Applicationdb.cs

```
using Microsoft.EntityFrameworkCore;
```

```
namespace EFCoreNCCSB.Models
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext() { }
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options) { }

        public DbSet<Officer> Officers { get; set; } // This represents tbl_officer
    }
}
```

Edit.cshtml

```
@model EFCoreNCCSB.Models.Officer
<h2>Edit Officer</h2>
<form asp-action="Edit">
    <input type="hidden" asp-for="Id" />
    <div class="form-group">
        <label>Name</label>
        <input asp-for="Name" class="form-control" />
    </div>
    <div class="form-group">
        <label>Gender</label>
        <input asp-for="Gender" class="form-control" />
    </div>
    <div class="form-group">
        <label>Phone</label>
        <input asp-for="Phone" class="form-control" />
    </div>
    <div class="form-group">
```

```
        <label>Department</label>
        <input asp-for="Department" class="form-control" />
    </div>
    <div class="form-group">
        <label>Position</label>
        <input asp-for="Position" class="form-control" />
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
</form>
```

Create.cshtml

```
@model EFCoreNCCSB.Models.Officer
<h2>Add Officer</h2>
<form asp-action="Create">
    <label>Name</label>
    <input asp-for="Name" class="form-control" />
    <span asp-validation-for="Name" class="text-danger"></span>
    <label>Gender</label>
    <input asp-for="Gender" class="form-control" />
    <span asp-validation-for="Gender" class="text-danger"></span>
    <label>Phone</label>
    <input asp-for="Phone" class="form-control" />
    <span asp-validation-for="Phone" class="text-danger"></span>
    <label>Department</label>
    <input asp-for="Department" class="form-control" />
    <span asp-validation-for="Department" class="text-danger"></span>

    <label>Position</label>
```

```
    <input asp-for="Position" class="form-control" />
    <span asp-validation-for="Position" class="text-danger"></span>

    <button type="submit" class="btn btn-success">Save</button>
</form>
```

Index.cshtml

```
@model IEnumerable<EFCoreNCCSB.Models.Officer>
<h2>Officers List</h2>
<a class="btn btn-success" asp-action="Create">Add Officer</a>
<table class="table">
    <thead>
        <tr>
            <th>Name</th>
            <th>Gender</th>
            <th>Phone</th>
            <th>Department</th>
            <th>Position</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var officer in Model)
        {
            <tr>
                <td>@officer.Name</td>
                <td>@officer.Gender</td>
                <td>@officer.Phone</td>
```

```
            <td>@officer.Department</td>
            <td>@officer.Position</td>
            <td>
                <a asp-action="Edit" asp-route-id="@officer.Id" class="btn btn-warning btn-sm">Edit</a>
                <a asp-action="Delete" asp-route-id="@officer.Id" class="btn btn-danger btn-sm">Delete</a>
            </td>
        </tr>
    }
  </tbody>
</table>
```

Delete.cshtml

```
@model EFCoreNCCSB.Models.Officer  // Single Officer

<h2>Delete Officer</h2>

<h3>Are you sure you want to delete this officer?</h3>
<div>
  <p><strong>Name:</strong> @Model.Name</p>
  <p><strong>Gender:</strong> @Model.Gender</p>
  <p><strong>Phone:</strong> @Model.Phone</p>
  <p><strong>Department:</strong> @Model.Department</p>
  <p><strong>Position:</strong> @Model.Position</p>

  <form asp-action="Delete" method="post">
    <input type="hidden" asp-for="Id" />
    <button type="submit" class="btn btn-danger">Delete</button>
```

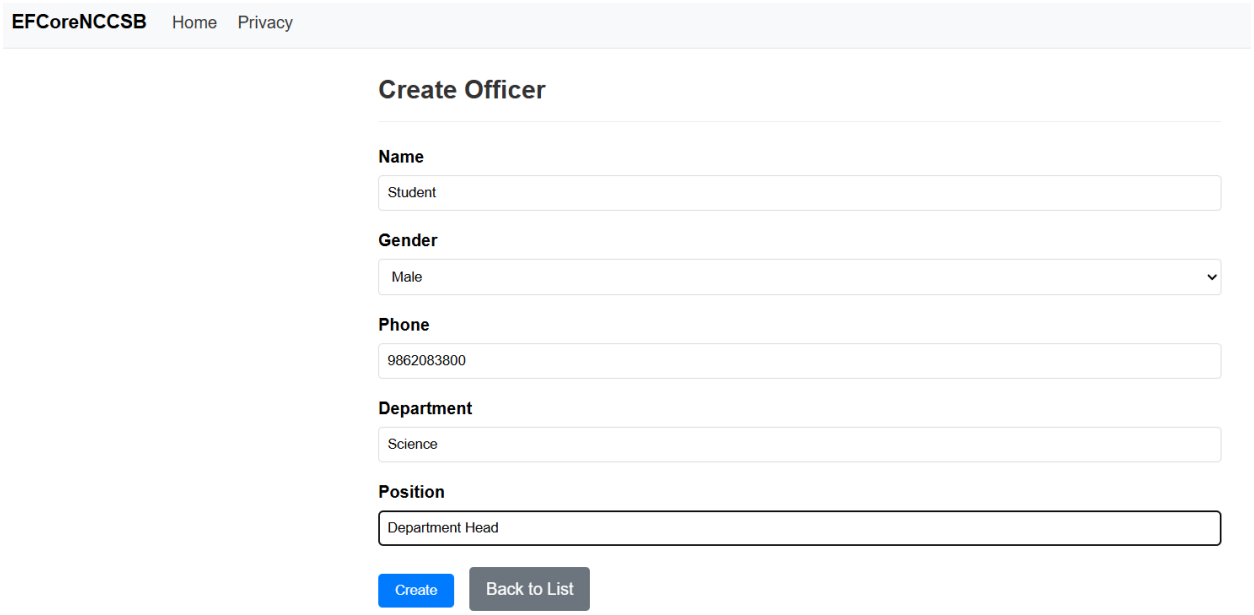
```
        <a asp-action="Index" class="btn btn-secondary">Cancel</a>
    </form>
</div>
```

Program.cs

```
using Microsoft.EntityFrameworkCore;
using EFCoreNCCSB.Models;
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
//adding configuration of sql server
builder.Services.AddDbContext<ApplicationDbContext>
    (options => options.UseSqlServer(builder.Configuration.GetConnectionString("cs")));
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();
```

OUTPUT:



22. Demonstrate different state management technique like SessionState, TempData, HttpContext

Controllers/StateController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace StateManagement1.Controllers
{
    public class StateController : Controller
    {
        public IActionResult SetSession()
        {
            HttpContext.Session.SetString("User", "Student");
            TempData["Message"] = "Session value has been set!";
            return RedirectToAction("GetSession");
        }

        public IActionResult GetSession()
        {
            string user = HttpContext.Session.GetString("User");
            ViewBag.User = user ?? "No session found!";
            ViewBag.Message = TempData["Message"];

            return View("SessionView");
        }

        public IActionResult TempDataDemo()
        {
```

```
        ViewData["data1"] = "this is data from view data (Student)";
        ViewBag.data2 = "this is data from view bag (CSIT)";
        TempData["data3"] = "data from temp data (ktm, tokha)";
        return View();
    }
  }
}
```

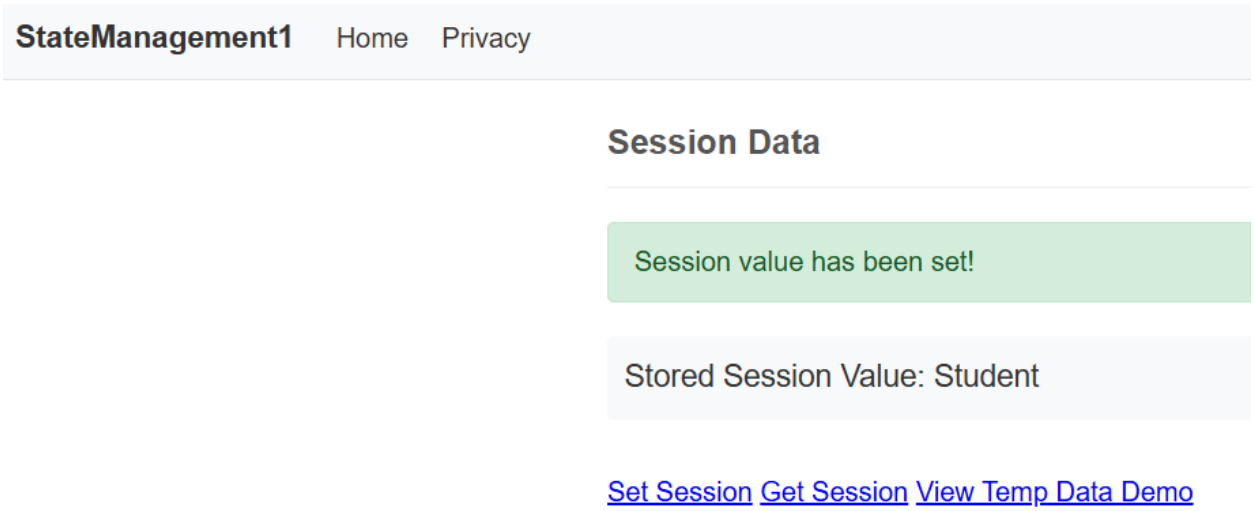
Views/State/SessionView.cshtml

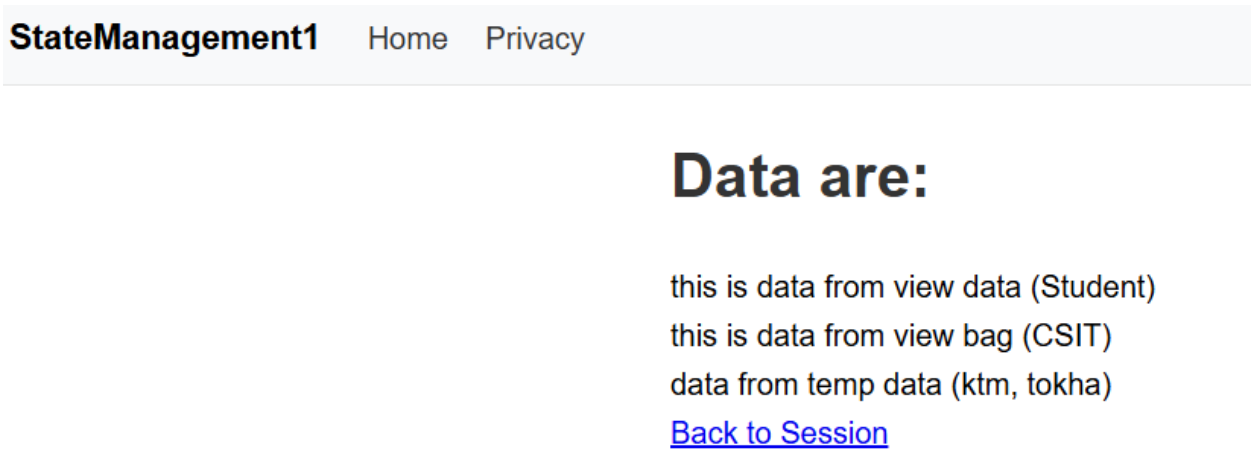
```
@{
  ViewData["Title"] = "Session Management";
}

<h2>Session Data</h2>

@if (ViewBag.Message != null)
{
  <p><strong style="color: green;">@ViewBag.Message</strong></p>
}

<p>Stored Session Value: <strong>@ViewBag.User</strong></p>

<a asp-controller="State" asp-action="SetSession">Set Session</a> |
<a asp-controller="State" asp-action="GetSession">Get Session</a> |
<a asp-controller="State" asp-action="TempDataDemo">View TempData Demo</a>
```

Views/TempDataDemo.cshtml

```
@{
    ViewData["Title"] = "TempData Demo";
}
<h2>Data are:</h2>
<h2>@ViewData["data1"]</h2>
<h2>@ViewBag.data2</h2>
<h2>@TempData["data3"]</h2>
<a asp-controller="State" asp-action="SetSession">Back to Session</a>
```

Output:





23. Demonstrate different client-side state management like cookies, Query string and hidden fields

Statecontrollerdemo.cs

```
using Microsoft.AspNetCore.Mvc;

namespace StateManagement1.Controllers
{
    public class StateControllerDemo : Controller
   {
        public IActionResult SetCookie()
        {
            CookieOptions option = new CookieOptions();
            option.Expires = DateTime.Now.AddMinutes(10);
            Response.Cookies.Append("UserName", "Student", option);
            ViewBag.Message = "Cookie has been set!";
            return View("ClientStateView");
        }

        // Get Cookie
        public IActionResult GetCookie()
        {
            string userName = Request.Cookies["UserName"];
            ViewBag.Message = userName ?? "No cookie found!";
            return View("ClientStateView");
        }
        public IActionResult QueryStringExample(string name, int age)
        {
            ViewBag.Message = $"Name: {name}, Age: {age}";
```

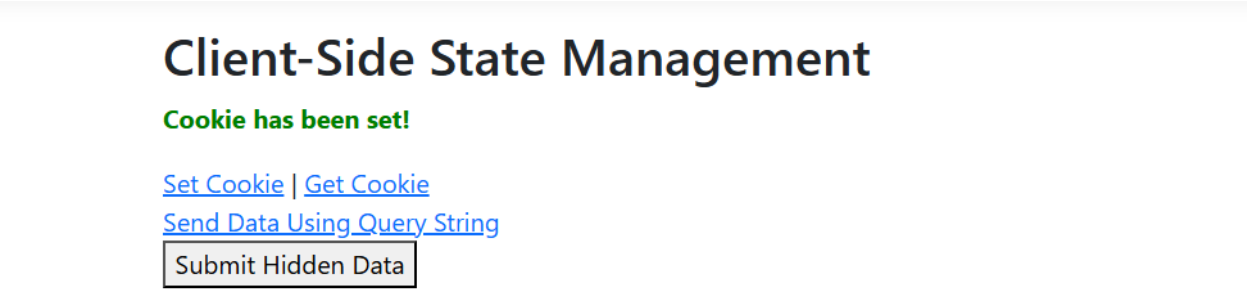
```
        return View("ClientStateView");
    }
    [HttpPost]
    public IActionResult SubmitHidden(string HiddenData)
    {
        ViewBag.Message = "Hidden Field Value: " + HiddenData;
        return View("ClientStateView");
    }

  }
}
```

Clientstate view.cshtml

```
@{
  ViewData["Title"] = "Client-Side State Management";
}

<h2>Client-Side State Management</h2>

@if (ViewBag.Message != null)
{
  <p><strong style="color: green;">@ViewBag.Message</strong></p>
}

<!-- Set & Get Cookies -->
<a asp-controller="State" asp-action="SetCookie">Set Cookie</a> |
<a asp-controller="State" asp-action="GetCookie">Get Cookie</a>
```

```
<!-- Query String Example -->
<br>
<a href="@Url.Action("QueryStringExample", "State", new { name = "Abhijit", age =55 })">
    Send Data Using Query String
</a>

<!-- Hidden Field Example -->
<form method="post" asp-action="SubmitHidden">
    <input type="hidden" name="HiddenData" value="SecretValue123" />
    <button type="submit">Submit Hidden Data</button>
</form>
```

OUTPUT:



24. Write a program to create complete form and validate using jquery and react.

src/component/FormComponent.jsx

```
import React, { useState } from 'react';
import $ from 'jquery';
const FormComponent = () => {
  const [formData, setFormData] = useState({
    name: '',

    email: '',
    password: '',
    rememberMe: false,
  });
  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setFormData({
      ...formData,
      [name]: type === 'checkbox' ? checked : value,
    });
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    $('.error').remove();
    let isValid = true;
    if (!formData.name) {
      isValid = false;
      $('#name').after('<span class="error">Name is required</span>');
    }
```

```
  if (!formData.email) {
    isValid = false;
    $('#email').after('<span class="error">Email is required</span>');
  } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
    isValid = false;
    $('#email').after('<span class="error">Email is invalid</span>');
  }
  if (!formData.password) {
    isValid = false;
    $('#password').after('<span class="error">Password is required</span>');
  }
  if (formData.password.length < 6) {
    alert('Password must be of at least 6 character');
  }
  if (isValid) {
    alert('Form submitted successfully!');
    // Here you can handle form submission, e.g., send data to the server
  }
};
return (
  <form onSubmit={handleSubmit}>
    <div>
      <label htmlFor="name">Name:</label>
      <input
        type="text"
        id="name"
        name="name"
        value={formData.name}
```

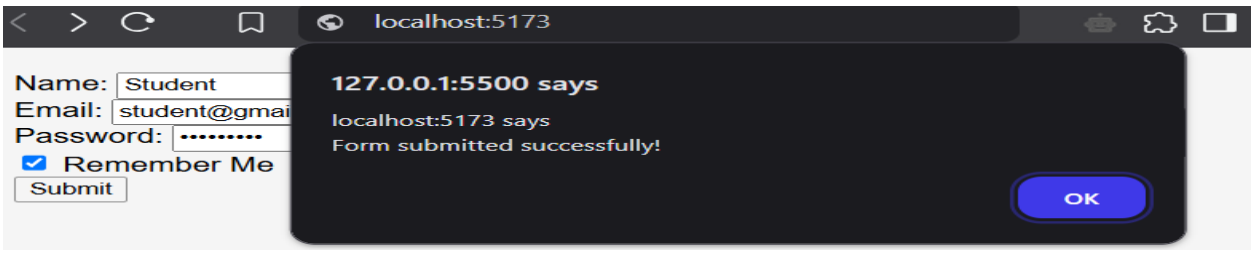
```
onChange={handleChange}
/>
</div>
<div>
  <label htmlFor="email">Email:</label>
  <input
    type="email"
    id="email"
    name="email"
    value={formData.email}
    onChange={handleChange}
  />
</div>
<div>
  <label htmlFor="password">Password:</label>
  <input
    type="password"
    id="password"
    name="password"
    value={formData.password}
    onChange={handleChange}
  />
</div>
<div>
  <label>
    <input
      type="checkbox"
      name="rememberMe"
```

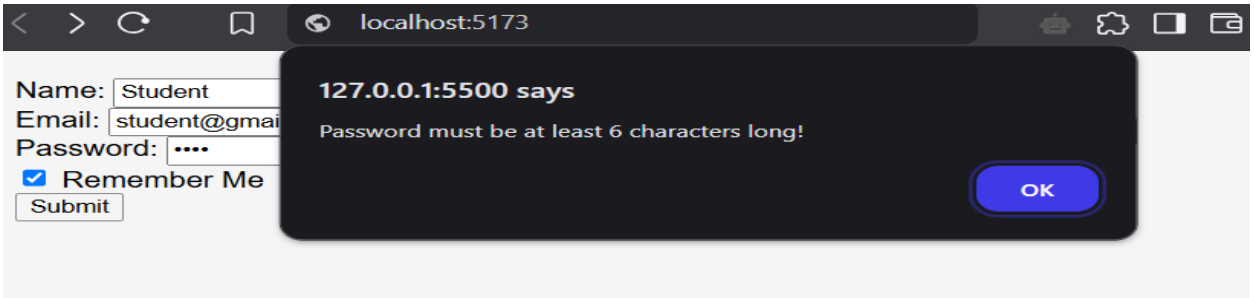
```
            checked={formData.rememberMe}
            onChange={handleChange}
          />
          Remember Me
        </label>
      </div>
      <button type="submit">Submit</button>
    </form>
  );
};
export default FormComponent;
```

App.jsx

```
import React from 'react';
import FormComponent from './component/FormComponent';
const App = () => {
  return (
    <div>
      <FormComponent />
    </div>
  ); };
export default App;
```

Output:





25. Write a program to demonstrate authentication and authorization (Role, claim and policies) by create a complete form in asp.net core.

Controllers/AuthenticationController.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
namespace AuthenticationAndAuthorizationDemo.Controllers
{
    public class AuthenticationController : Controller
    {
        [HttpGet]
        public IActionResult Login(string returnUrl = "/")
        {
            ViewData["returnUrl"] = returnUrl;
            return View();
        }
        [HttpPost]
        public async Task<IActionResult> Login(string uname, string pass, string returnUrl = "/")
        {
            if (uname == "Student" && pass == "1234")
            {
                var claims = new List<Claim>
                {
                    new Claim(ClaimTypes.NameIdentifier, uname),
                    new Claim(ClaimTypes.Name, uname),
                    new Claim(ClaimTypes.Role, "Admin"),
```

```
            new Claim("CanEdit", "true") // Custom claim
        };
        var claimsIdentity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
        var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);
        await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
claimsPrincipal);
        return RedirectToAction("Dashboard"); // Redirect to Dashboard after successful login
      }
      ViewData["Error"] = "Invalid username or password";
      return View();
    }
    [Authorize]
    public IActionResult Dashboard()
    {
      return View("Dashboard");
    }
    [Authorize(Roles = "Admin")]
    public IActionResult AdminPage()
    {
      return View("AdminPage");
    }
    [Authorize(Policy = "CanEditPolicy")]
    public IActionResult EditPage()
    {
      return View("EditPage");
    }
    [Authorize(Policy = "IsStudent”)]
    public IActionResult SpecialPage()
```

```
        {
            return View("SpecialPage");
        }
        [HttpPost]
        public async Task<IActionResult> Logout()
        {
            await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            return RedirectToAction("Login");
        }
    }
}
```

Views/Authentication/Login.cshtml

```
@{
    string retUrl = ViewData["returnUrl"] as string ?? "/";
}
<h1>Login</h1>
@if (ViewData["Error"] != null)
{
    <p style="color:red">@ViewData["Error"]</p>
}
<form method="post" action="@Url.Action("Login",
"Authentication")?returnUrl=@System.Net.WebUtility.UrlEncode(retUrl)">
    <label>Username:</label>
    <input type="text" name="uname" required /><br />
    <label>Password:</label>
    <input type="password" name="pass" required /><br />
    <input type="submit" value="Login" />
</form>
```

Views/Authentication/Dashboard.cshtml

```
@{
    ViewData["Title"] = "Dashboard";
}
<h2>Welcome, @User.Identity.Name!</h2>
<ul>
    <li><a href="@Url.Action("AdminPage", "Authentication")">Go to Admin Page</a></li>
    <li><a href="@Url.Action("EditPage", "Authentication")">Go to Edit Page</a></li>
    <li><a href="@Url.Action("SpecialPage", "Authentication")">Go to Special Page</a></li>
    <li>
        <form method="post" action="@Url.Action("Logout", "Authentication")">
            <button type="submit">Logout</button>
        </form>
    </li>
</ul>
```

Views/Authentication/AdminPage.cshtml

```
<h1>Admin Page</h1>
<p>Only Admins can access this page.</p>
```

Views/Authentication/EditPage.cshtml

```
<h1>Edit Page</h1>
<p>Only users with the "CanEdit" claim can access this page.</p>
```

Views/Authentication/SpecialPage.cshtml

```
<h1>Special Page</h1>
<p>Only Student can access this page.</p>
```

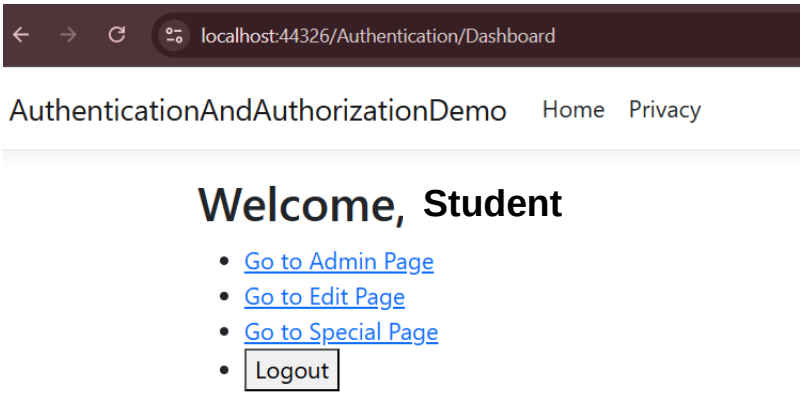
Program.cs

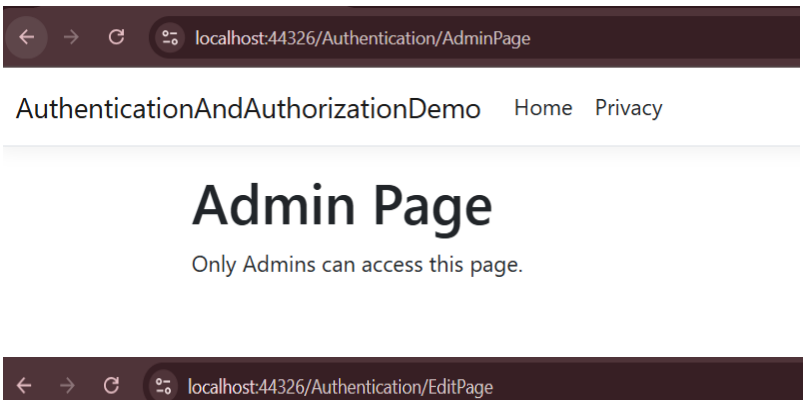
```
using Microsoft.AspNetCore.Authentication.Cookies;
```

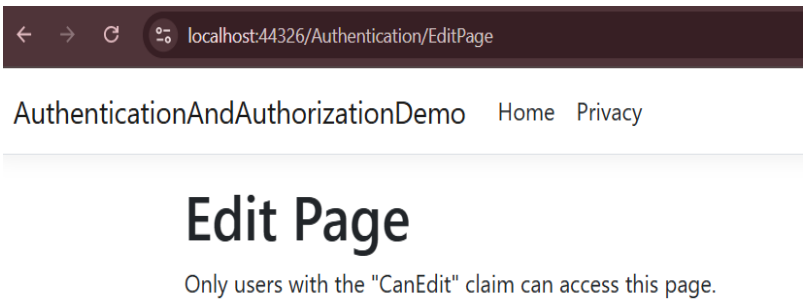
```
using System.Security.Claims;

var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddControllersWithViews();
// Adding authentication mechanism: cookie
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(x =>
    {
        x.LoginPath = "/Authentication/Login"; // Specify the correct Login path here
    });

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("CanEditPolicy", policy =>
        policy.RequireClaim("CanEdit", "true"));

    options.AddPolicy("IsStudent", policy =>
        policy.RequireClaim(ClaimTypes.Name, "Student"));
});
var app = builder.Build();
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}
app.UseHttpsRedirection();
```

```
app.UseStaticFiles();


app.UseRouting();
app.UseAuthentication(); // Add authentication middleware here
app.UseAuthorization();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Authentication}/{action=Login}/{id?}");
app.Run();
```

OUTPUT:







26. Write a program to prevent SQLInjectionAttack, Cross Site Request forgery (CSRF) and open redirect attack.

Code:

Safecontroller.cs

```
using System;
using System.Web;
using Microsoft.AspNetCore.Mvc;
namespace SecureApp.Controllers
{
    public class SafeController : Controller
   {
        [HttpGet]
        public IActionResult SecureForm()
        {
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]  // CSRF Protection
        public IActionResult SecureForm(string userInput)
        {
            if (string.IsNullOrWhiteSpace(userInput))
            {
                ViewBag.Message = "Input cannot be empty.";
                return View();
            }
            string safeInput = HttpUtility.HtmlEncode(userInput);

            ViewBag.Message = "Processed Input: " + safeInput;
            return View();
```

```
        }
        public IActionResult SafeRedirect(string returnUrl)
        {
            Uri redirectUri;
            if (Uri.TryCreate(returnUrl, UriKind.RelativeOrAbsolute, out redirectUri))
            {
                if (!redirectUri.IsAbsoluteUri || redirectUri.Host == Request.Host.Host) // Only allow internal redirects
                {
                    return Redirect(returnUrl);
                }
            }

            return RedirectToAction("SecureForm"); // Safe default redirection
        }
    }
}
```

Secureform.cshtml

```
@{
    ViewData["Title"] = "Secure Form";
}
<h2>Secure Form</h2>
<!-- Form to prevent CSRF & XSS -->
<form method="post" action="/Safe/SecureForm">
    @Html.AntiForgeryToken() <!-- CSRF Token -->
    <label>Enter Text:</label>
    <input type="text" name="userInput" required>
    <button type="submit">Submit</button>
</form>
```

```
@if (ViewBag.Message != null)
{
    <p style="color: green">@ViewBag.Message</p>
}
<hr>
<!-- Open Redirect Prevention -->
<h3>Try Unsafe Redirect</h3>
<form action="/Safe/SafeRedirect" method="get">
    <input type="text" name="returnUrl" placeholder="Enter redirect URL">
    <button type="submit">Click to Try Unsafe Redirect</button>
</form>
```

Output: