Lab 18: ASP.NET Core MVC Application

Title:

Create an ASP.Net Core MVC project (WebApp1ByPrakash) with Razor pages, Model validation, and CRUD forms.

Theory:ASP.NET Core MVC follows Model-View-Controller pattern. Models represent data with validation attributes. Views (Razor pages) display UI. Controllers handle HTTP requests and return responses. Tag Helpers simplify form creation. Server-side validation ensures data integrity before processing.

Packages Installed:

Microsoft.AspNetCore.Mvc (for MVC functionality)

Microsoft.EntityFrameworkCore (optional if you are using database operations)

System.ComponentModel.DataAnnotations (for model validation)

WebAppByMohit

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <RootNamespace>WebApp_by_Mohit</RootNamespace>
  </PropertyGroup>

</Project>
```

#Student Controller.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using WebAppByMohit.Models;


namespace WebAppByMohit.Controllers
{
    public class StudentsController : Controller
    {
        static List<Student> students = new List<Student>();

        public IActionResult Index()
        {
            return View(students);
```

```
        }

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

        [HttpPost]
        public IActionResult Create(Student student)
        {
            if (ModelState.IsValid)
            {
                students.Add(student);
                return RedirectToAction("Index");
            }
            return View(student);
        }
    }
}
```

Student Model.cs

```
using System.ComponentModel.DataAnnotations;

namespace WebAppByMohit.Models
{
    public class Student
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Name is required")]
        public string? Name { get; set; }

        [Range(1, 200)]
        public int RollNo { get; set; }

        [Required(ErrorMessage = "College is required")]
        public string? College { get; set; }
    }
}
```

View

```
// Create view.cshtml
@model WebAppByMohit.Models.Student

<h2>Add Student</h2>
```

```
<form asp-action="Create">
    Name: <input asp-for="Name" /> <br />
    <span asp-validation-for="Name"></span><br />

    Roll No: <input asp-for="RollNo" /> <br />
    <span asp-validation-for="RollNo"></span><br />

    College: <input asp-for="College" /> <br />
    <span asp-validation-for="College"></span><br />

    <button type="submit">Save</button>
</form>

<hr />
<p>
    Name: Mohit Tharu <br />
    Roll No: 117 <br />
    College: Patan Multiple Campus
</p>



//index.cshtml
@model IEnumerable<WebAppByMohit.Models.Student>

<h2>Student List</h2>

<table border="1">
    <tr>
        <th>Name</th>
        <th>Roll No</th>
        <th>College</th>
    </tr>

    @foreach (var s in Model)
    {
        <tr>
            <td>@s.Name</td>
            <td>@s.RollNo</td>
            <td>@s.College</td>
        </tr>
    }
</table>

<br />
<a href="/Students/Create">Add New Student</a>
```

#program.cs

```
var builder = WebApplication.CreateBuilder(args);

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

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.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

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


app.Run();
```

#Output

Add Student

Name:

Roll No:

College:

Save

Name: Mohit Tharu
Roll No: 117
College: Patan Multiple Campus

19. Create a project to illustrate dependency injection in ASP.Net Core. (If your name is Shyam hen project name should be "WebApp2ByShyam")

Theory:

Dependency Injection is a design pattern where objects receive dependencies from external sources. ASP.NET Core has a built-in DI container with three service lifetimes: Transient (new instance every time), Scoped (one instance per request), Singleton (single instance for app lifetime).

 # WebApp2ByMohit

```
<h2>Dependency Injection Demo</h2>

<p>@ViewBag.Message</p>

<hr />

<p>
        Name: Mohit Tharu <br />
        Roll No: 117 <br />
        College: Patan Multiple Campus
</p>
```

#Controller

```
//homeController.cs
using Microsoft.AspNetCore.Mvc;
using WebApp2ByMohit.Services;

namespace WebApp2ByMohit.Controllers
{
  public class HomeController : Controller
  {
    private readonly IMyService _service;

    // Constructor Injection
    public HomeController(IMyService service)
```

```
        {
            _service = service;
        }

        public IActionResult Index()
        {
            ViewBag.Message = _service.GetMessage();
            return View();
        }
    }
}
```

#Service

```
using Microsoft.AspNetCore.Mvc;
using WebApp2ByMohit.Services;

namespace WebApp2ByMohit.Controllers
{
    public class HomeController : Controller
    {
        private readonly IMyService _service;

        // Constructor Injection
        public HomeController(IMyService service)
        {
            _service = service;
        }

        public IActionResult Index()
        {
            ViewBag.Message = _service.GetMessage();
            return View();
        }
    }
}
```

//IMyservice.cs

```
using Microsoft.AspNetCore.Mvc;
using WebApp2ByMohit.Services;
```

```
namespace WebApp2ByMohit.Controllers
{
   public class HomeController : Controller
   {
      private readonly IMyService _service;

      // Constructor Injection
      public HomeController(IMyService service)
      {
         _service = service;
      }

      public IActionResult Index()
      {
         ViewBag.Message = _service.GetMessage();
         return View();
      }
   }
}
```

```
//Myservice.cs
namespace WebApp2ByMohit.Services
{
   public class MyService : IMyService
   {
      public string GetMessage()
      {
         return "Hello from Dependency Injection Service";
      }
   }
}
```

#program.cs

```
using WebApp2ByMohit.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
```

```
builder.Services.AddControllersWithViews();



builder.Services.AddTransient<IMyService, MyService>();


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.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

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


app.Run();
```

#output

Dependency Injection Demo

Hello from Dependency Injection Service

Name: Mohit Tharu
Roll No: 117
College: Patan Multiple Campus