MVC-Day8

Routing

⚠ `Routing` **is a process of mapping the URL to the controller and action method**

`app.MapControllerRoute`

- is a middleware that used to map the request URL to the controller and action method
- URL has to match the `pattern` to be mapped to the controller

  ⚠ **we can change the order and add more parts from the** `pattern`

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

    //Url: /Home/Index  => id is optional
```

`pattern`

```
pattern: "{action}/{controller}/{id}"
//Url: /Index/Home/1 => id is mandatory
```

- we can add more parts to the pattern name

```
pattern: "{controller=Home}/{action=Index}/{id?}/{name?}"
//Url: /Home/Index/1/John => id and name are optional
```

```
pattern: "{controller}/{action=Index}/{id?}"
//Url: /Home/Index/1 => id is optional if action is not provided it will take
Index as default
```

- we can add static parts to the pattern

```
pattern: "Admin/{controller=Home}/{action=Index}/{id?}"
//Url: /Admin/Home/Index/1 => id is optional
```

- we can add constraints to the pattern (id should be a integer)

```
pattern: "{controller=Home}/{action=Index}/{id:int?}"
//Url: /Home/Index/1 => id is optional and should be a integer
```

- `max` :works with numbers

```
pattern: "{controller=Home}/{action=Index}/{id:max(100)}"
//Url: /Home/Index/1 => id is optional and should be less than 100
```

- there are other constraints like `min` , `range` , `alpha` , `regex` etc

Route Attribute

`[Route("xyz")]`

- when added to action method it will override the default routing
- then the only way to access the action method is by using the URL `xyz`

```
[Route("xyz")]
public IActionResult Index()
{
    return View();
}
```

we have to use `/xyz` to access the action method instead of `/Home/Index`

✓ **we can use multiple route attributes to the same action method**

```
[Route("xyz")]
[Route("Department/Index")]
```

```
//Url: /xyz or /Department/Index
public IActionResult Index()
{
    return View();
}
```

we can change only the controller name using route attribute

```
[Route("Hissen/{action}")]
//Url: /Hissen/Index or /Hissen/About

//we can also add default value to the action
[Route("Hissen/{action = Index}")]
// Url: /Hissen => Index


[Route("Hissen/{action=Index}/{id?}")]
// Url: /Hissen/Index/1 => id is optional
public DepartmentController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

we use route attribute mostly with Web API

- in MVC we use convention-based routing more

Razor Pages

`Razor Pages`

- create new project with (ASP.NET Core Web Application(Razor Pages))
- Razor Pages are similar to MVC but with less complexity

has simiralities with MVC in terms of folder structure

`wwwroot` , `appsettings.json` etc

- no Models, Views, Controllers folders
- instead we have `Pages` folder
- each page has a `.cshtml` file and a `.cshtml.cs` file
- `.cshtml` file is the view and `.cshtml.cs` file is the controller
- `@page` directive is used to define the route of the page(Must so we can route to the page directly)
- `_ViewStart.cshtml` : is used to define the layout of the page
- `_ViewImports.cshtml` : is used to define the namespaces that are used in the pages

  `@addTagHelper` is used to add the tag helpers to the page(like `asp-action` , `asp-controller` etc)

  - `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers` is used to add all the tag helpers

```
//startup.cs
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    //rest of the code

    app.MapRazorPages();// instead of MapControllerRoute
}
```

`@page` directive

- has to be the first line in the `.cshtml` file to be able to route to the page directly through the URL

```
// Show.cshtml
@page
@{
    int x = 10;
}
```

```
<h1>Show Page</h1>
<p>Value of x is @x</p>
```

create new folder student with `Index.cshtml`

- route is `/student/index` or `/student` as default behavior the `Index` page is loaded

> ⚠ **if we change the name of the page to `Show` then the route will be `/student/show` and `/student` will not work**

split the page into two files `Index.cshtml` and `Index.cshtml.cs`

- view file is `Index.cshtml`, controller file is `Index.cshtml.cs`
- using `add razor page` option in the `Pages` folder
- add page named `Display`
- we will have `DisplayModel` that inherits from `PageModel`
- `PageModel` is similar to `Controller` in MVC
- `OnGet` method is used to handle the get request
- `OnPost` method is used to handle the post request
- `OnGet` and `OnPost` are similar to `HttpGet` and `HttpPost` in MVC

```
//Display.cshtml.cs
public class DisplayModel : PageModel
{
    Public int X { get; set; } = 10;//this alone will show the value of X in the
view using @Model.X
    public IActionResult OnGet()
    {
        X = 20;//change the value of X
        //as it is a get request and this method is will be called
    }

    public IActionResult OnPost()
    {
        return Page();
    }
}
```

```
//Display.cshtml @page @model DisplayModel //display the value of X
<h1>Display Page</h1>
<p>Value of X is @Model.X</p>
```

✓ **we can do everything that we do in MVC in Razor Pages**

- Razor Pages has `ViewData` , `TempData` , `ViewBag` etc
- Razor Pages has `Tag Helpers` like `asp-action` , `asp-controller` etc

make post request to the page

- add a form to the page

```
<!--
    this will post the form to the same page in the URL
 -->
<form method="post">
  <input type="text" name="x" />
  <input type="submit" value="Submit" />
</form>
```

edit the method `OnPost` to get the value of `x` from the form

```
public IActionResult OnPost(int x)//model binding
{
    X = x;//this will change the value of X to the value of x from the form
}
```

`[BindProperty]` attribute

- used to bind the property to the form
- by default support only post request
- to make it support get request we have to add `SupportsGet = true`

```
[BindProperty]
//this will make model binder to bind the value of x to the property X comming from
the post request
[BingProperty(SupportsGet = true)]
//this will support get request
public int X { get; set; } = 10;
```

we can add other Properties to the model and bind them to the form

```
[BindProperty]
public int Y { get; set; } = 20;
```

```
<form method="post">
  <input type="text" name="x" />
  <input type="text" name="y" />
  <input type="submit" value="Submit" />
</form>
```

✓ **we can add binder for all the properties in the model**

```
[BindProperties]
//this will bind all the properties in the model to the values comming from the
form
public class DisplayModel : PageModel
{
    public int X { get; set; } = 10;
    public int Y { get; set; } = 20;
}
```

we can return a `Page` or `RedirectToPage`

```
//Display.cshtml.cs
public IActionResult OnGet(int x)
{
    X = x;
    return Page();
    //this will return the same page
}
public IActionResult OnPost(int x)
```

```
{
    X = x;
    return RedirectToPage("Display");
    //this will redirect to the same page
}
```

`[BindNever]` attribute

- used to exclude the property from the model binding

add models

- create a folder named `Models`
- create a folder named `Repos`, or `Services`
- we can `reverse engineer` the database to create the models and the context from (PowerTools)
- generate the models and the context from the database
- we choose student and department tables

`[InverseProperty]` attribute

- used to define the relationship between the models when they 2 relationships between them
- used to define the navigation property in the other model
- `[InverseProperty("Students")]` in the `Student` model

`partial class`

- so we can add code like override methods to the model
- and on generating the models again the code will not be lost
- we can add partial class to the model and add the code to it

metadata type

- used to add metadata to the model

- used to add validation to the model
  -...etc

```
//StudentMetadata.cs
//now add [MetadataType(typeof(StudentMetadata))] to the Student model
[MetadataType(typeof(StudentMetadata))]
//now properties in the StudentMetadata will be applied to the Student model
public partial class Student
{

}
public class StudentMetadata
{
    [MinLength(3)]
    //now Name Property in the Student model will have a minimum length of 3
    public string Name { get; set; }
}
```

Break

using last lab repos

- creating `Department` folder => it is the controller
- add razor page `Index` to the `Department` folder
- `Index.cshtml` and `Index.cshtml.cs` is view and action method

```
//Index.cshtml.cs
public class IndexModel : PageModel
{

    public IDeptRepo deptRepo;
    public IndexModel(IDeptRepo _deptRepo)
    {
        deptRepo = _deptRepo;
    }
    public List<Department> Departments { get; set; }
    public void OnGet()
    {
        Departments = deptRepo.GetAll();
    }

}
```

```
//Index.cshtml @page @model IndexModel @{ }

<table class="table table-bordered">
  <thead>
    <tr>
      <th>Id</th>
      <th>Name</th>
      <th>Capacity</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var dept in Model.Departments) {
    <tr>
      <td>@dept.DeptId</td>
      <td>@dept.DeptName</td>
      <td>@dept.Capacity</td>
    </tr>
    }
  </tbody>
</table>
```

add `Create` page

- add `Create` page to the `Department` folder
- `Create.cshtml` and `Create.cshtml.cs` is view and action method

```
//Create.cshtml.cs
public class CreateModel : PageModel
{
    public IDeptRepo deptRepo;
    public CreateModel(IDeptRepo _deptRepo)
    {
        deptRepo = _deptRepo;
    }
    [BindProperty]
    public Department Department { get; set; }
    public void OnGet()
    {
        Department = new Department();
    }

    public IActionResult OnPost()
    {
        if(!ModelState.IsValid)
        {
            return Page();
        }
        deptRepo.AddDepartment(Department);
        return RedirectToPage("Index");
    }
}
```

```
//Create.cshtml @page @model CreateModel @{ }

<!--
    default will post to the same page
 -->
<form method="post">
  <!--
        we can specify the page to post to
      -->
  <!-- <form method="post" asp-page="Create"> -->
  <div class="form-group">
    <label asp-for="Department.DeptName"></label>
    <input asp-for="Department.DeptName" class="form-control" />
  </div>
  <div class="form-group">
    <label asp-for="Department.Capacity"></label>
    <input asp-for="Department.Capacity" class="form-control" />
  </div>
  <input type="submit" value="Create" class="btn btn-primary" />
</form>
```

✓ add `Details` page

- add `Details` page to the `Department` folder
- `Details.cshtml` and `Details.cshtml.cs` is view and action method
- to force the page to take the id from the URL we have to add `@page "{id}"` to the `.cshtml` file

```
//Details.cshtml
<!-- @page "{id}"  -->
@page "{id?}"
<!-- to make the id optional -->

@model DetailsModel @{ }

<h1>Details Page</h1>
<p>Id: @Model.Department.DeptId</p>
<p>Name: @Model.Department.DeptName</p>
<p>Capacity: @Model.Department.Capacity</p>
```

```
//Details.cshtml.cs

public class DetailsModel : PageModel
{
    public IDeptRepo deptRepo;
    public DetailsModel(IDeptRepo _deptRepo)
    {
        deptRepo = _deptRepo;
    }
    public Department Department { get; set; }
    public void OnGet(int? id)
    //as we added {id} to the page directive we have to add id to the method
    //? to make it optional if it is not provided
    {
        if(id == null){
            return BadRequest();
        }

        Department = deptRepo.GetById(id);
        if(Department == null){
            return NotFound();
        }

    }
}
```

✓ **add link to details page**

```
<!-- Index.cshtml -->
<!-- rest of the code  -->
@foreach (var dept in Model.Departments) {
```

```
<tr>
  <td>@dept.DeptId</td>
  <td>@dept.DeptName</td>
  <td>@dept.Capacity</td>
  <td>
    <a asp-page="Details" asp-route-id="@dept.DeptId">Details</a>
  </td>
</tr>
}
```

we can do everything we did in MVC in Razor Pages

we can use `Scaffold` to generate the Controller and the Views from the model (MVC)

bind only specific properties from method parameters

- `[Bind("DeptName, Capacity")] Department department`
- this will bind only `DeptName` and `Capacity` from the input to the `Department` object

we use `Scaffold` to generate the pages in the Razor Pages

---

lab

#lab

-