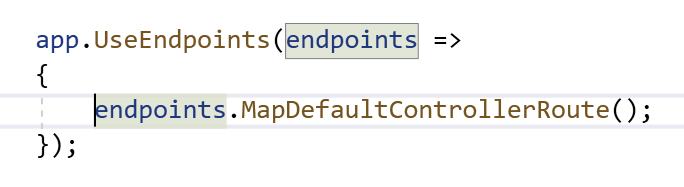
**Agenda**

* Attribute Routing
* Logging
* Model Validation
* File Provider
* File Upload
* Async Controllers
* Working with multiple environment
* Using Developer Exception Page
* Configure Status Code Pages

**Attribute Routing**

* Primary advantage of attribute based routing is that it allows you to define your routes in the same file as the controller.
* You can use Route attribute to specify the route for specific action.
* All the routes in the controller class should start with the same prefix.
* You can also set common prefix for the entire controller class.

1. Create AuthController
2. Change Startup.cs



1. Add Following code into that



1. Check the Output



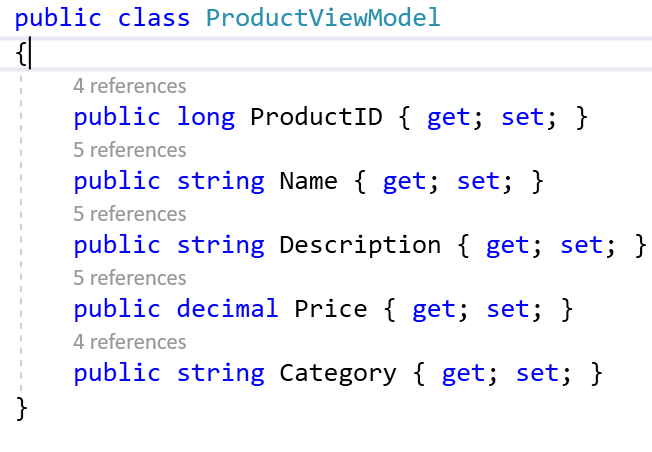
1. You can also define routes at the Controller level



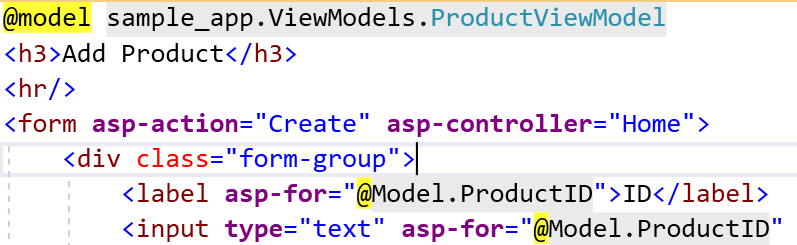
1. Check the Output

**Model Validation**

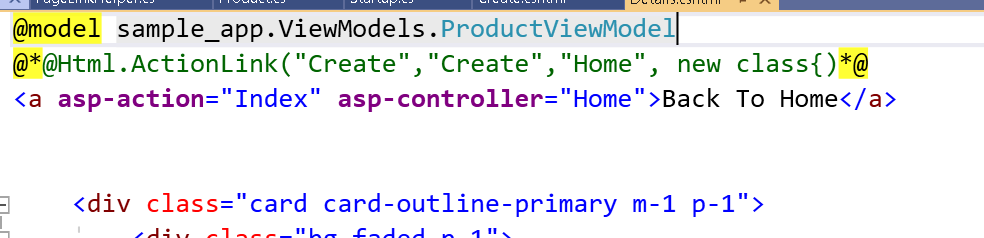
1. Don’t add attributes on models directly
2. Create ViewModel class for holding such validation. It means , My View should attached to ViewModel not to Model.



1. Make The Changes in the code to Make it work with this new model.
2. Change Create.cshtml



1. Change Details.cshtml



1. Convert the code into appropriate viewmodels

[HttpPost]

public IActionResult Create(ProductViewModel productViewModel)

{

Product product = new Product

{

ProductID = productViewModel.ProductID,

Category = productViewModel.Category,

Description=productViewModel.Description,

Name=productViewModel.Name,

Price =productViewModel.Price

};

\_repository.AddProduct(product);

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult Details(int id)

{

var product = \_repository.GetProduct(id);

ProductViewModel productViewModel = new ProductViewModel

{

ProductID =product.ProductID,

Category = product.Category,

Description = product.Description,

Name = product.Name,

Price = product.Price

};

return View(productViewModel);

}

1. Check the Output
2. Add Validation Attributes

public class ProductViewModel

{

[Required(ErrorMessage = "ProductID is Required Field")]

public long ProductID { get; set; }

[Required(ErrorMessage = "Name is Required Field")]

[Display(Name = "Product Name")]

public string Name { get; set; }

[Required(ErrorMessage = "Description is Required Field")]

[StringLength(20, ErrorMessage = "{0} length must be between {2} and {1}.", MinimumLength = 5)]

public string Description { get; set; }

[Required(ErrorMessage = "Price is Required Field")]

[Range(0, 999)]

public decimal Price { get; set; }

[Required(ErrorMessage = "Category is Required Field")]

public string Category { get; set; }

}

1. Change the Create.cshtml

<form **asp-action**="Create" **asp-controller**="Home">

<div class="form-group">

<label **asp-for**="@Model.ProductID">ID</label>

<input **type**="text" **asp-for**="@Model.ProductID" class="form-control" placeholder="ID">

<span **asp-validation-for**="@Model.ProductID" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Name">Name</label>

<input **type**="text" **asp-for**="@Model.Name" class="form-control" placeholder="Name">

<span **asp-validation-for**="@Model.Name" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Price">Price</label>

<input **type**="text" **asp-for**="@Model.Price" class="form-control" placeholder="Price">

<span **asp-validation-for**="@Model.Price" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Category">Category</label>

<input **type**="text" **asp-for**="@Model.Category" class="form-control" placeholder="Category">

<span **asp-validation-for**="@Model.Category" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Description">Description</label>

<input **type**="text" **asp-for**="@Model.Description" class="form-control" placeholder="Description">

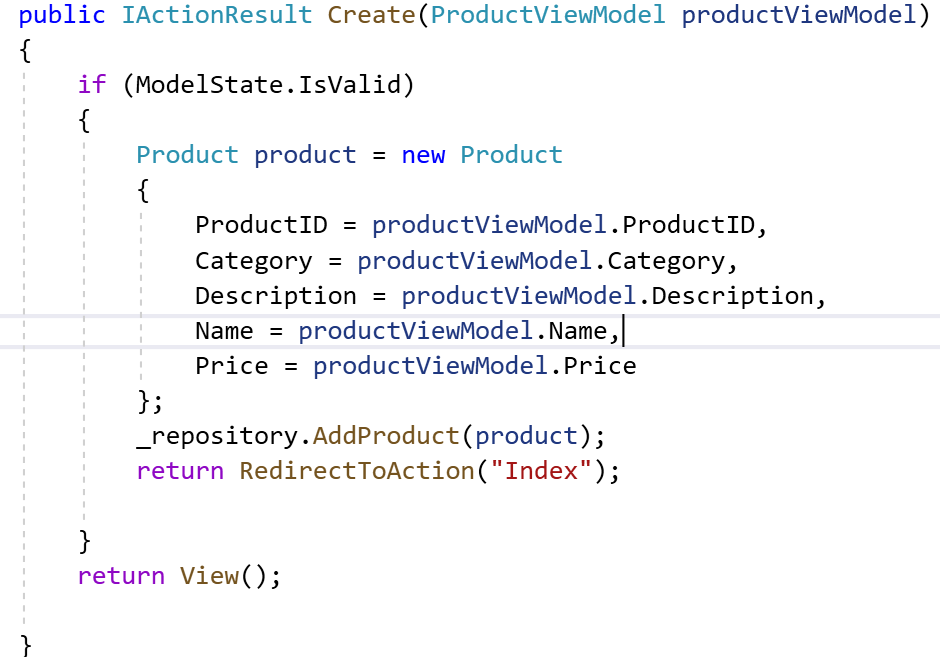
<span **asp-validation-for**="@Model.Description" class="text-danger"></span>

</div>

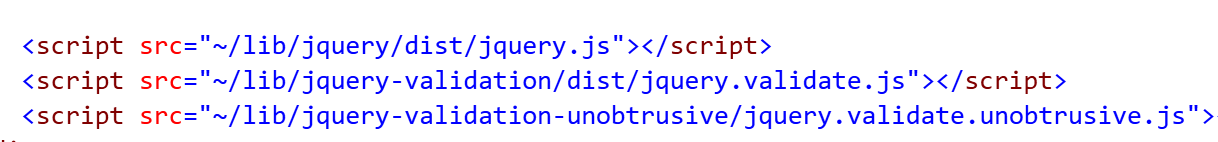
<button type="submit" class="btn btn-default">Submit</button>

</form>

1. Add ModelState code



1. IF you notice , Currently nothing is working , as we have to make it work at client side. We need to add jquery library in Layout.cshtml .



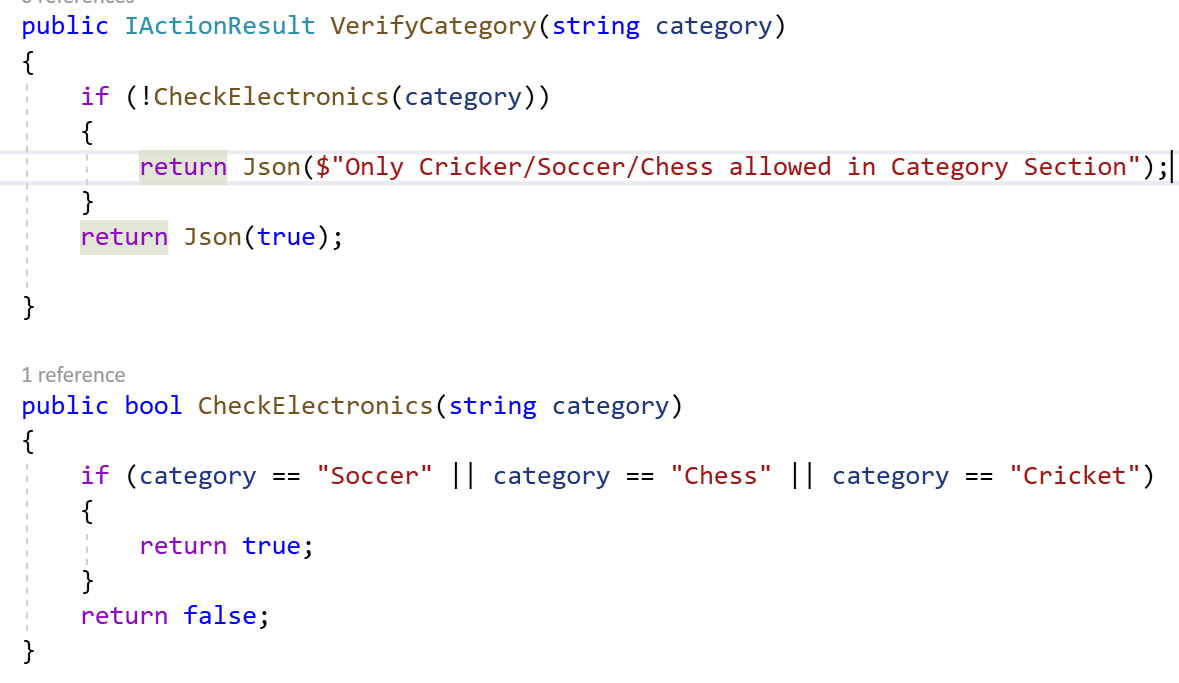
1. Check the Output

**Remote Validation**

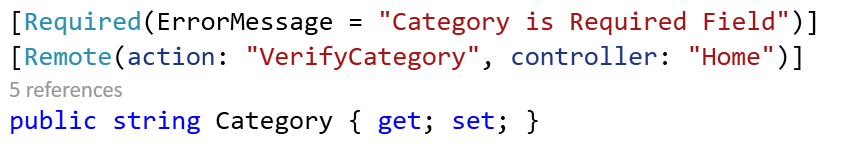
* The [Remote] attribute implements client-side validation that requires calling a method on the server to determine whether field input is valid.

The jQuery Validation [remote](https://jqueryvalidation.org/remote-method/) method expects a JSON response:

* true means the input data is valid.
* false, undefined, or null means the input is invalid. Display the default error message.
  1. Add Following code in HomeController



* 1. Apply Remote Attribute to Category Section



How to add Custom Attribute : If there is scenario which doesn’t provide by the Built in attributes we can create custom attributes

Differnce between Remote and Custom Attribute is

 With validating the data in the controller action, we can't reuse it throughout the application and much more.

Refer [The Complete Guide to Validation in ASP.NET MVC 3 - Part 2](http://www.devtrends.co.uk/blog/the-complete-guide-to-validation-in-asp.net-mvc-3-part-2) for a good article

* Add New Attribute that is MfgDate in the Model

1. Go to Infrastructure folder and add New Attribute

public class ValidMfgDateAttribute : ValidationAttribute

{

public int Year { get; }

public string GetErrorMessage() =>

$"Mfg Date Should not be less than Current Year";

protected override ValidationResult IsValid(object value,

ValidationContext validationContext)

{

var product = (ProductViewModel)validationContext.ObjectInstance;

var currentYear = DateTime.Now.Year;

if (product.MfgDate.Year < currentYear)

{

return new ValidationResult(GetErrorMessage());

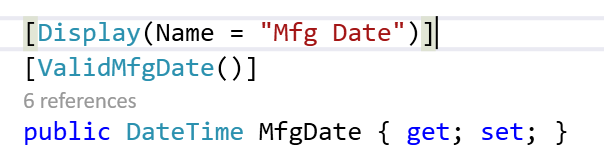
}

return ValidationResult.Success;

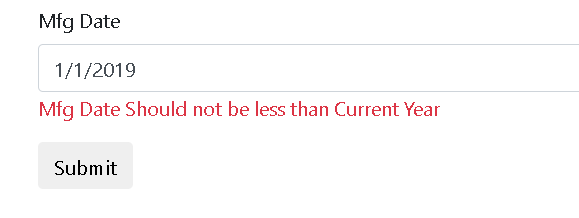
}

}

1. Apply it On Model



1. Check the Output

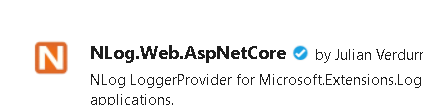


**Logging**

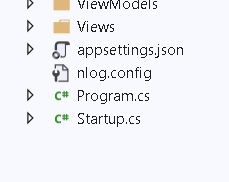
* ASP.NET Core doesn't include a logging provider for writing logs to files. To write logs to files from an ASP.NET Core app, consider using a [third-party logging provider](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-3.1#third-party-logging-providers). (<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-3.1#bilp>)

NLog is an open source and flexible framework that works with various .NET platforms including .NET Standard, Full framework (i.e., .NET Framework 4.7), Xamarin (Android and iOS), Windows phone, and UWP. NLog is easy to use and extend. Also, it provides more flexibility in term of configuration. We can also change the logging configuration on-the-fly. The target is used to store, display, and pass the log messages to the provided destination. NLog can write a log to one or more targets at the same time. NLog provides more than 30 targets that include file, event log, database, email, console, etc.

1. Add Library Through Nugets



1. Add nlog.config file



1. Add a code

<?xml version="1.0" encoding="utf-8" ?>

<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<!-- the targets to write -->

<targets>

<!-- write to file -->

<target name="applog" xsi:type="File"

fileName="D:\FreeLancerAssignments\BOA\sample-app\logs\applog-${shortdate}.log"

layout="${longdate} - ${message} -

${exception:format=StackTrace}${newline}" />

</targets>

<!-- rules to map from logger name to target -->

<rules>

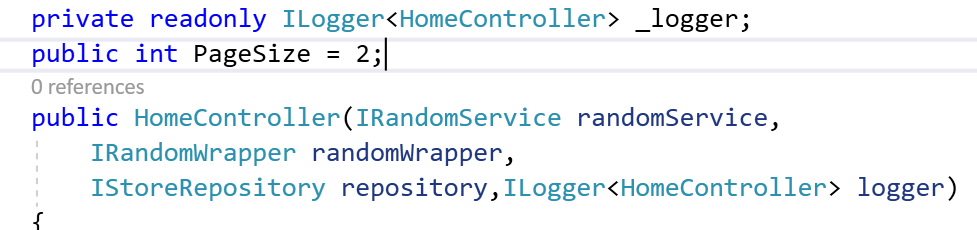
<!-- all logs including Microsoft -->

<logger name="\*" minlevel="Trace" writeTo="applog" />

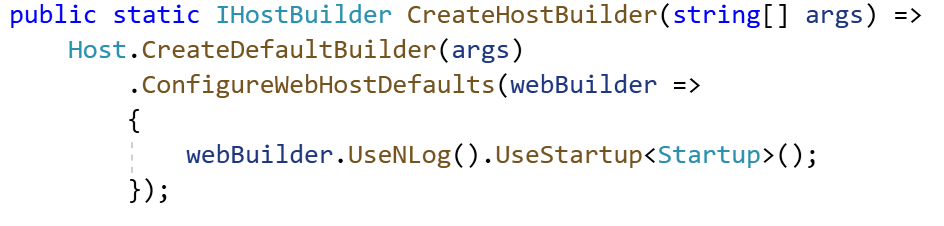
</rules>

</nlog>

1. Go to Controller and add Logging Support



1. Add NLog Support



1. Check the Output

**File Providers in ASP.Net Core**

* ASP.Net Core Abstracts file System through the use of File Providers.
* Internally Some Component use File Providers to Get the Files

1. WebHostingEnvironment : Get apps content root and web root
2. Static Files : Locate Statuc files
3. Razor : locate Page and views

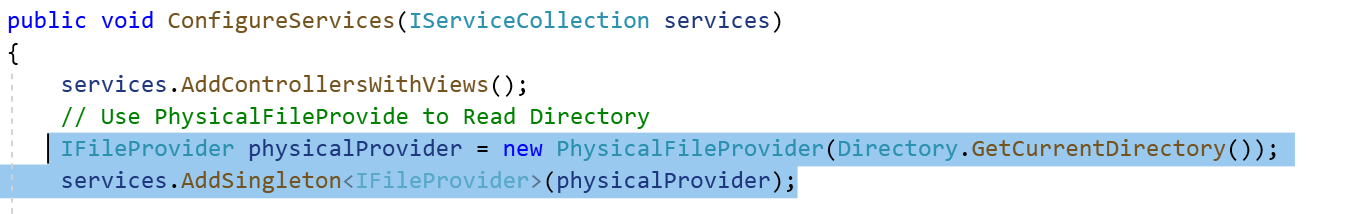
PhysicalFileProvider that wraps the System.IO File type and provides the access to Physical file system. Its able to access all the paths to a directory and subdirectory.

Embedded File Provider : Use to access files that are embedded in assemblies

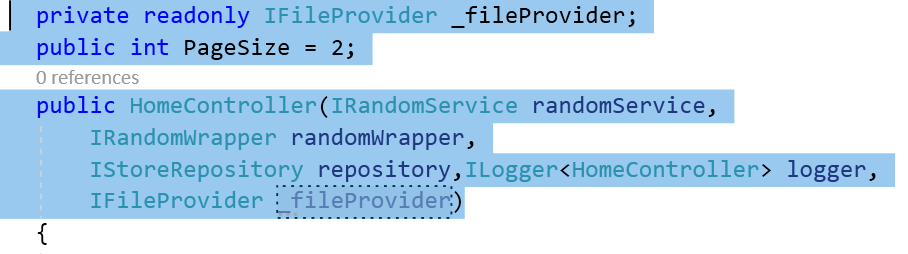
Composite File Provider : Use to provide combine access to the files from one or more providers.

Demo :

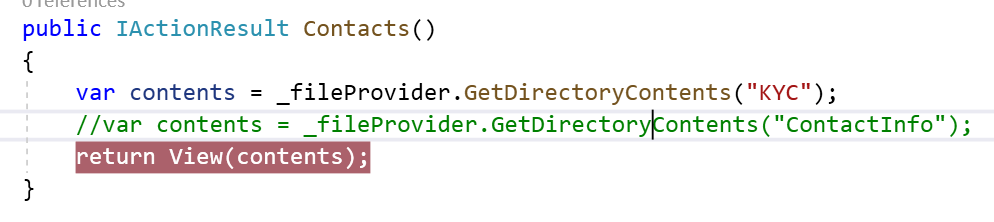
1. Add a Support for File Provider in ConfigureServices



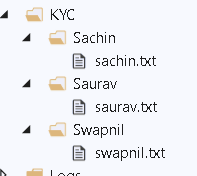
1. DI FileProvider in HomeController



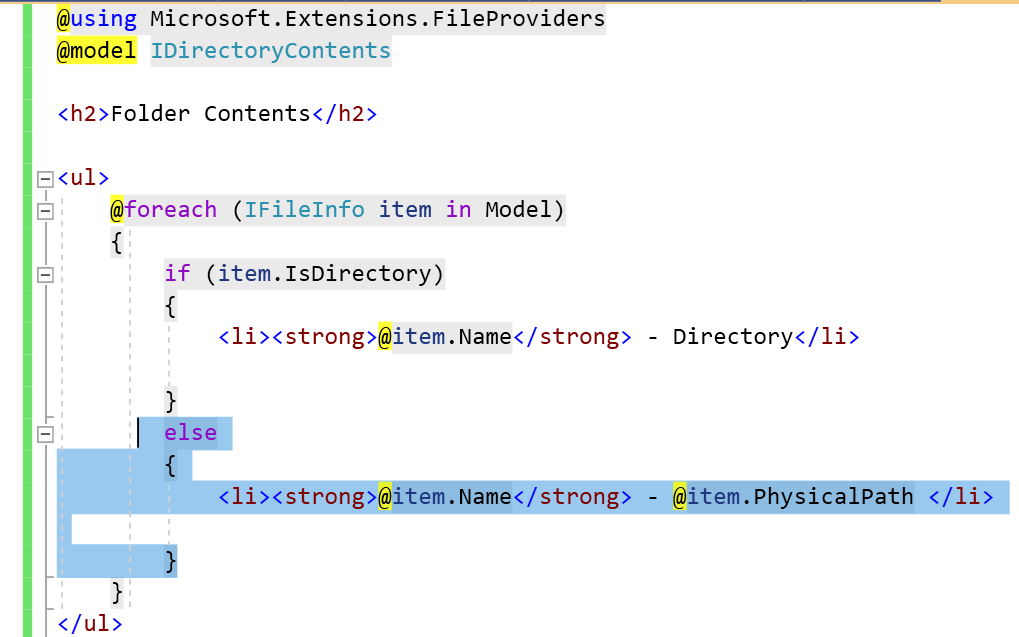
1. Add New Action that can Get the KYC Dir information



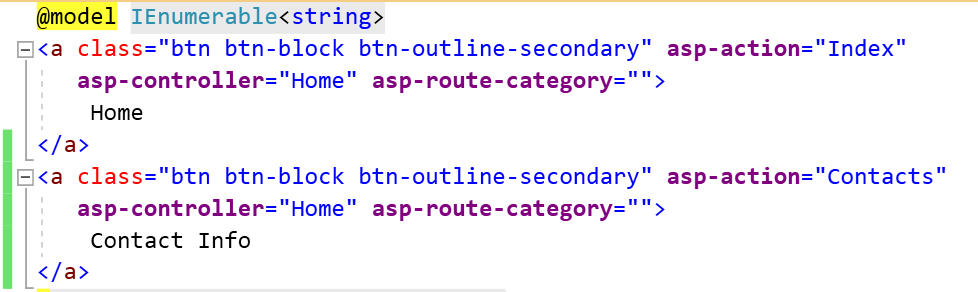
1. Create Dir KYC Whose information we want to extract



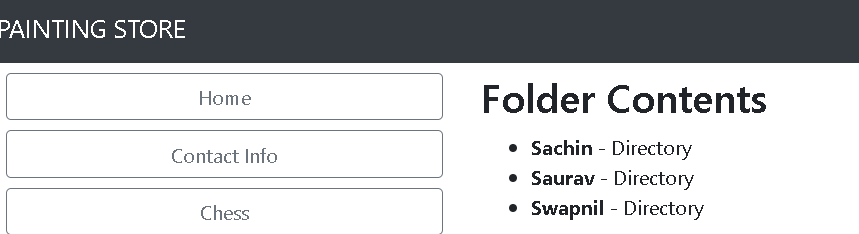
1. Display the content of directory on Contact Page.



1. Add Link for ContactInfo Or KYC in ViewComponents NavigationMenu



1. Check the Output



**Upload File in .Net Core**

1. Add a File Upload Controls in Contact.html

<form method="post" enctype="multipart/form-data" **asp-controller**="Home" **asp-action**="FileUpload">

<div class="form-group">

<div class="col-md-10">

<p>Upload one or more files using this form:</p>

<input type="file" name="files" multiple />

</div>

</div>

<div class="form-group">

<div class="col-md-10">

<input type="submit" value="Upload" />

</div>

</div>

</form>

1. Add a Logic to Upload multiple files in Specific Folder

[HttpPost("FileUpload")]

public async Task<IActionResult> FileUpload(List<IFormFile> files)

{

long size = files.Sum(f => f.Length);

var filePaths = new List<string>();

foreach (var formFile in files)

{

if (formFile.Length > 0)

{

var path = Path.Combine(Directory.GetCurrentDirectory(),"KYC",formFile.FileName);

filePaths.Add(path);

using (var stream = new FileStream(path, FileMode.Create))

{

await formFile.CopyToAsync(stream);

}

}

}

// process uploaded files

// Don't rely on or trust the FileName property without validation.

return Ok(new { count = files.Count, size, filePaths });

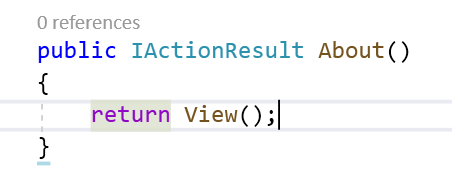
}

1. Check the Output

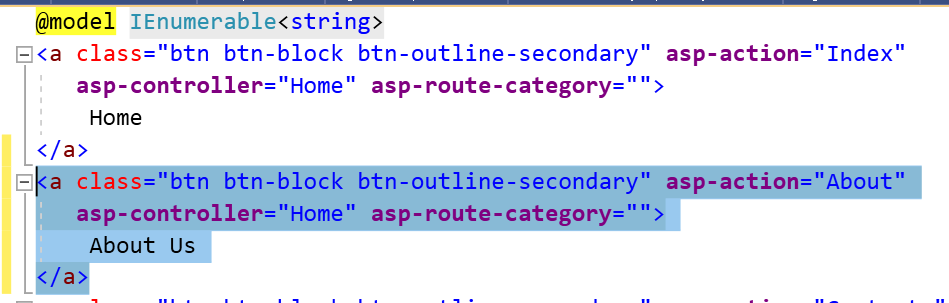
**Using Developer Exception Page**

Lab : How to work with Developer Exception Page

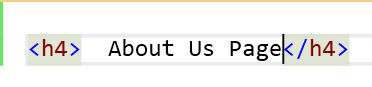
1. Add About Action Method In Home Controller



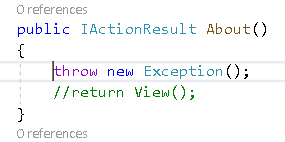
1. Add Link for the same using NavigationMenu Component



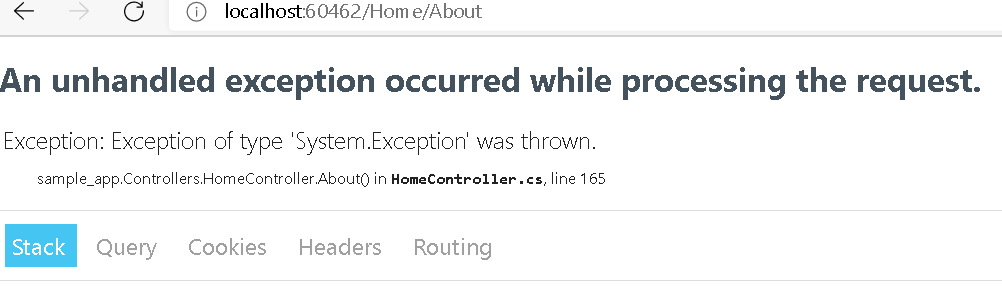
1. Add View For the same



1. Check the Output
2. Add Exception code in About Page

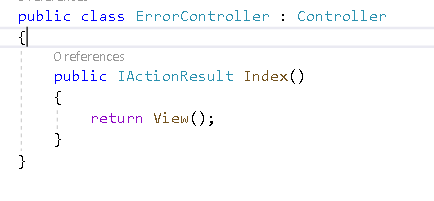


1. Check the Output after hitting About Page

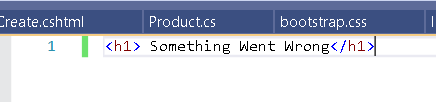


Problem : User Should not be able to see this Page. So we need to make sure that proper message should be displayed to User instead of Exception handling related information.

1. Add Controller for Error Handling



1. Add Index Page for this Error



1. Add a Conditional Code for Exceptions



1. Change the Setting from launchSettings



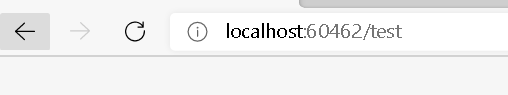
1. Check the Output

**Status Code Pages**

* An addition method of handling erros is UseStatusCodePages middleware
* Provides basic handling for various errors that may occur by displaying short explanation of the error code to the user
* Its designed to handle cases where some part of the request went wrong without throwing an exception
* This method will capture html status codes between 400 to 599

Lab : Demo :

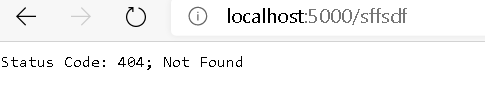
1. Type Some Wrong Url : No Output



1. Add Status Code Middleware in Configure Method



1. Type some wrong url



Areas

* ASP.Net Feature use to organize related functionality into group as a separate.
* Provide a way to Partition an ASP.Net Core Web app into Smaller functional groups.

Functional Groups

1. Own Set of Razor Pages
2. Controllers
3. Views
4. Models

Real World Example

* Ecommerce app with multiple business unit such as

1. Checkout
2. Billing
3. Search

* Naukri.com : Where we have 3 area
  + 1. Employer
    2. Administrator
    3. JobSeeker

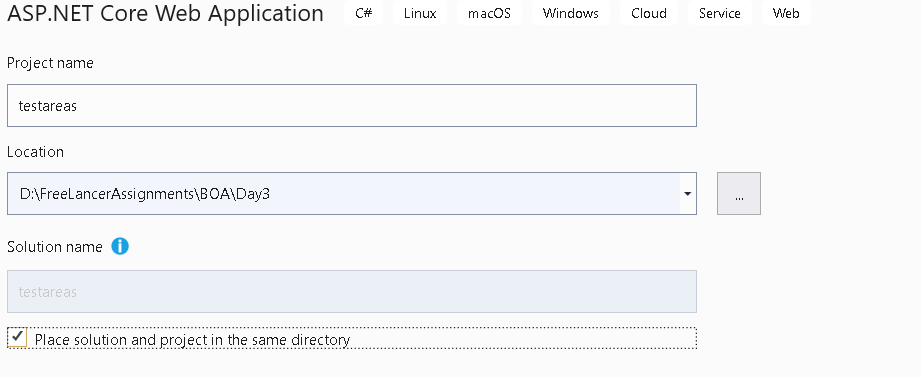
Have their own area to contain views, controller , Razor pages and models.

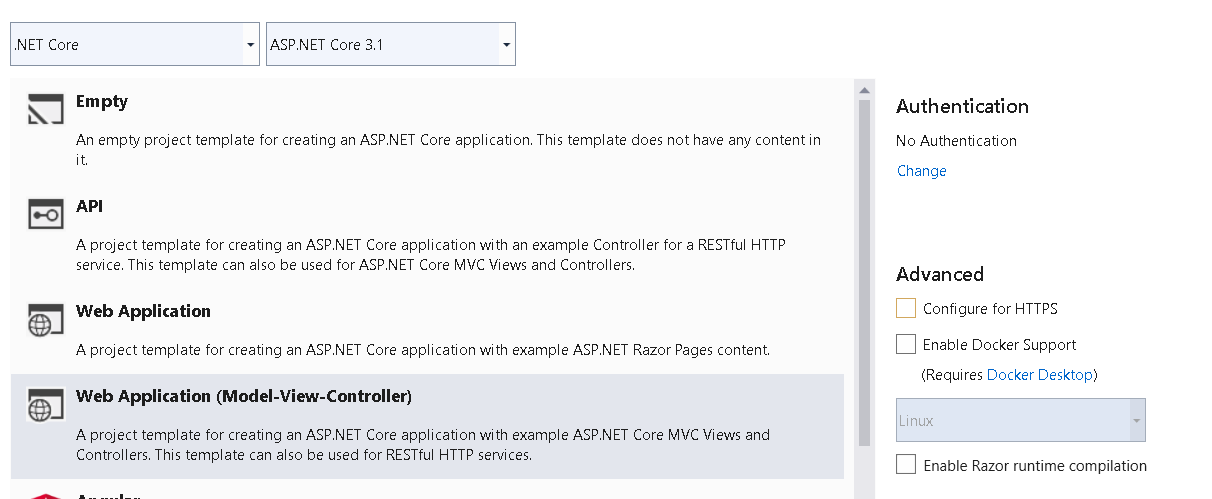
When to Go for Areas

1. App made of multiple high level functional components that can be logically separated
2. Partition app so that each functional area can be worked independently.

Demo : How to use Areas

1. Create New MVC Project

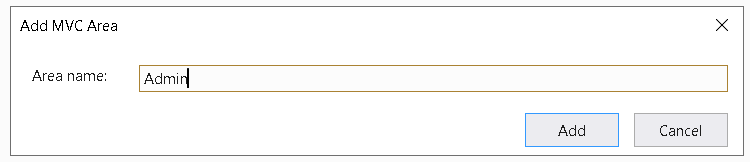




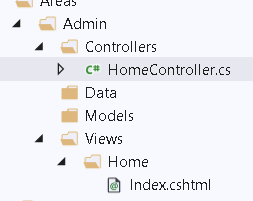
1. Add New Folder Areas



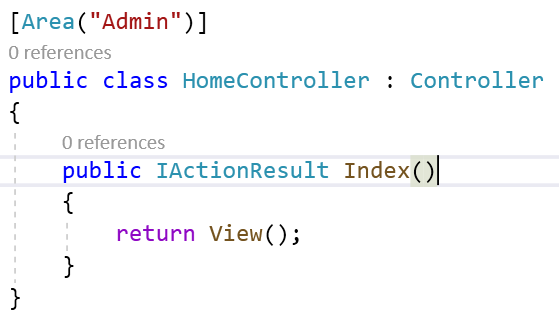
1. Right Click on Areas Folder and Add new Area : Give Name to it as Admin



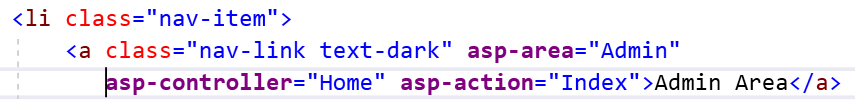
1. Add Controller and Views for Admin Area



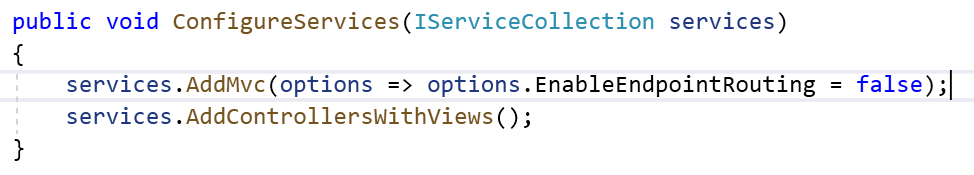
1. we have to tell MVC which controller is associated with which area. We can do this thing by decorating controller with "Area" attribute.



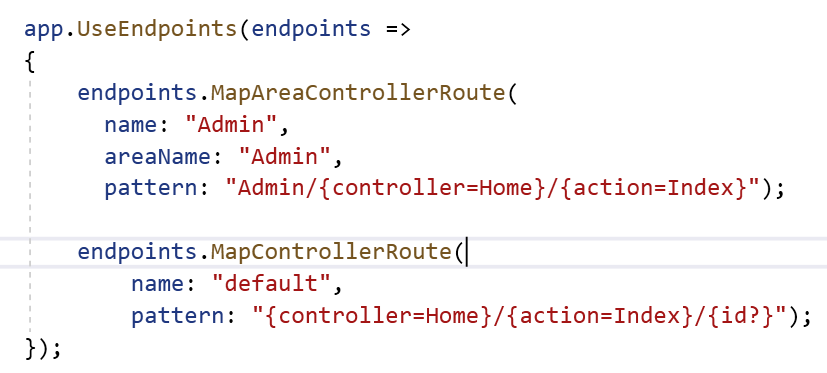
1. Go to Main area and change the Home Link



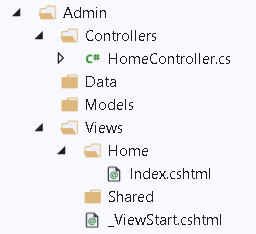
1. Area is the MVC concept, for enabling areas in .Net Core, we have to register MVC . And EnableEndpointRouting is set to false in the startup.cs file.



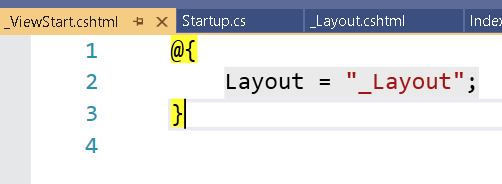
1. Add a Routing Support For admin



1. Check the Output
2. Add \_ViewStart.cshtml in the Admin area



1. \_viewStart.cshtml contains following code



1. Add Employer Area using same steps.