Exception Handling in ASP.NET Core MVC

In this article, we will be discussing various ways of handling an exception in ASP.NET Core MVC.

Introduction

For ASP.NET Core MVC, we have similar situation or discussion, but, with major differences:

1. We will not discuss the Try-Catch-Finally approach, because it is language related issue;
2. Due to Exception Filter, the approach is just secondary importance in ASP.NET Core app, we will just make brief discussion at the end.

* A: Exception Handling in Development Environment for ASP.NET Core MVC
  + UseDeveloperExceptionPage
* B: Exception Handling in Production Environment for ASP.NET Core MVC
  + Approach 1: UseExceptionHandler
    - 1: Exception Handler Page
    - 2: Exception Handler Lambda
  + Approach 2: UseStatusCodePages
    - 1: UseStatusCodePages, and with format string, and with Lambda
    - 2: UseStatusCodePagesWithRedirects
    - 3: UseStatusCodePagesWithReExecute
  + Approach 3: Exception Filter
    - Local
    - Global

A: Exception Handling in Developer Environment

The ASP.NET Core starup templates generate the following code,

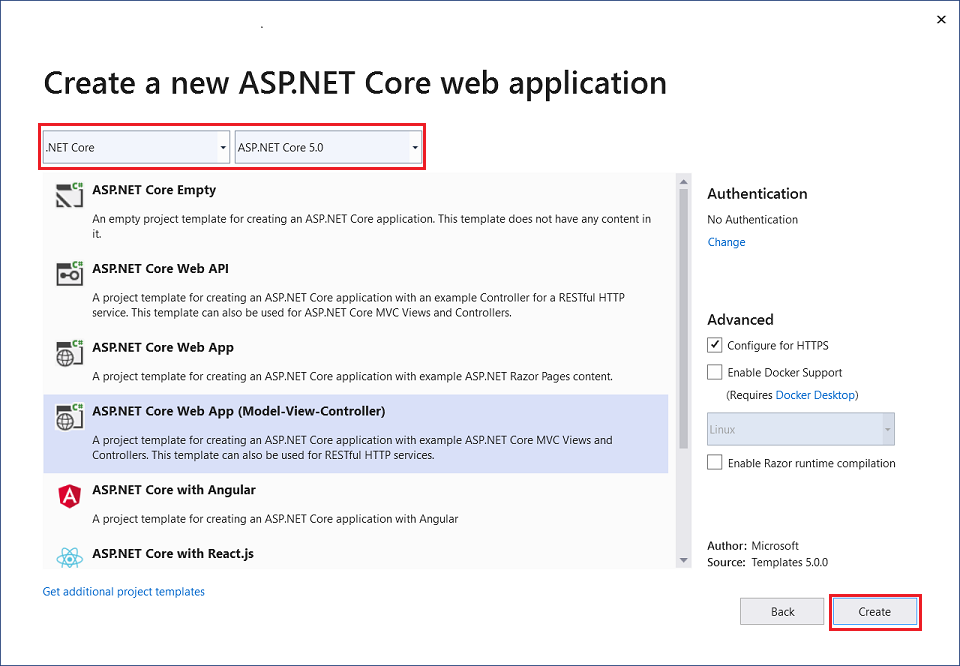
1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. ......
9. }

The UseDeveloperExceptionPage extension method adds middleware into the request pipeline. The *Developer Exception Page* displays developer friendly detailed information about request exceptions. This helps developers in tracing errors that occur during development phase.  
  
As this middleware displays sensitive information, it is advisable to add it only in development environment. The developer environment is a new feature in .NET Core. We will demostrate this below.

**Step 1 - Create an ASP.NET Core MVC application**

We use the current version of Visual Studio 2019 16.8 and .NET 5.0 SDK to build the app.

1. Start Visual Studio and select Create a new project.
2. In the Create a new project dialog, select ASP.NET Core Web Application > Next.
3. In the Configure your new project dialog, enter *ErrorHandlingSample* for Project name.
4. Select Create.
5. In the Create a new ASP.NET Core web application dialog, select,  
     
   1. .NET Core and ASP.NET Core 5.0 in the dropdowns.
   2. ASP.NET Core Web App (Model-View-Controller).
   3. Create



**Step 2 - Change code in Home Controller**

Replace the Index method in the HomeController with the code below:

1. **public** IActionResult Index(**int**? id = **null**)
2. {
3. **if** (id.HasValue)
4. {
5. **if** (id == 1)
6. {
7. **throw** **new** FileNotFoundException("File not found exception thrown in index.chtml");
8. }
9. **else** **if** (id == 2)
10. {
11. **return** StatusCode(500);
12. }
13. }
14. **return** View();
15. }

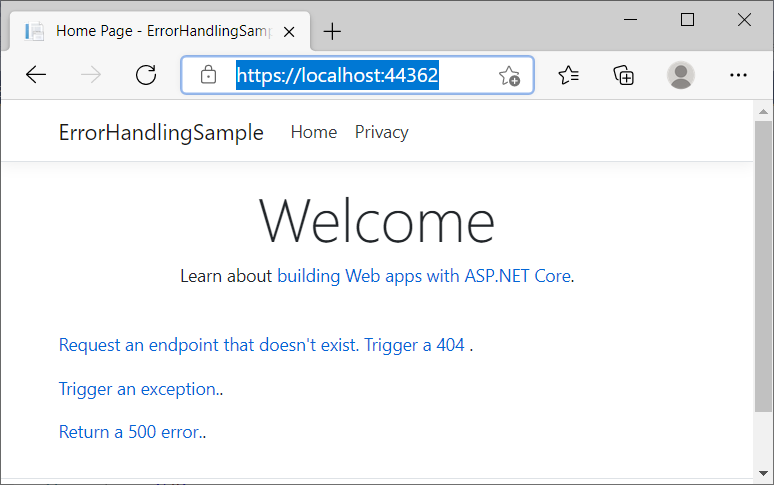
**Step 3 - Change code in Index view**

Add the code in the bottom of Home/Index view, i.e., the file Index.cshtml in Views/home directory,

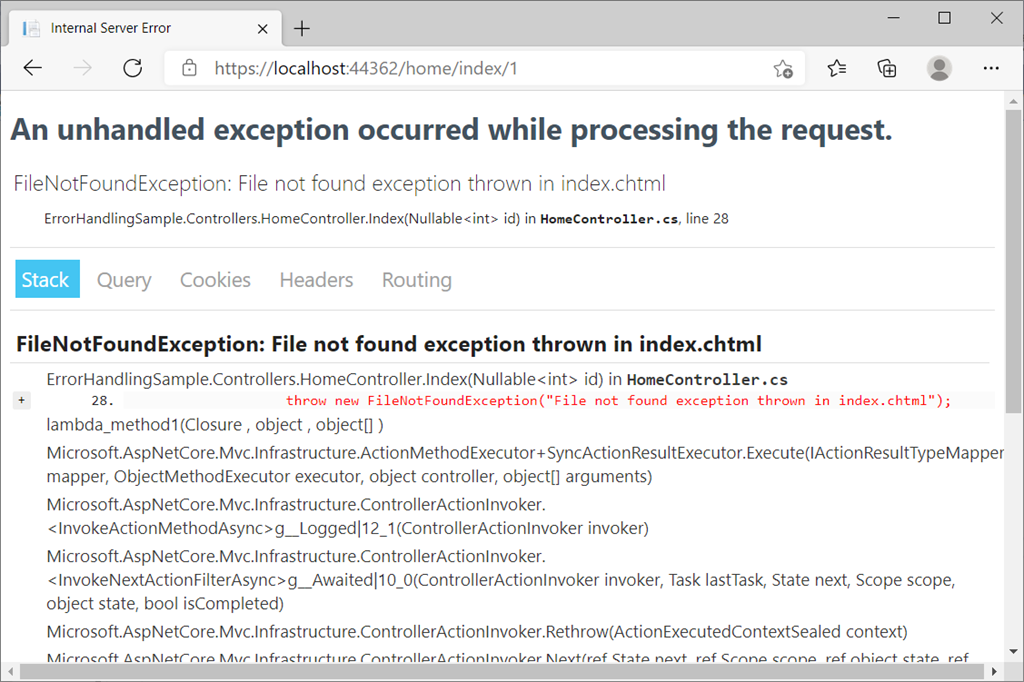
1. <br />
3. <div **class**="text-left">
4. <p>
5. <a href="/NoSuchPage">
6. Request an endpoint that doesn't exist. Trigger a 404
7. </a>.
8. </p>
9. <p><a href="/home/index/1">Trigger an exceptionn</a>.</p>
10. <p><a href="/home/index/2">Return a 500 error.</a>.</p>
11. </div>

**Step 4 - Run app and Test**

Run the app,



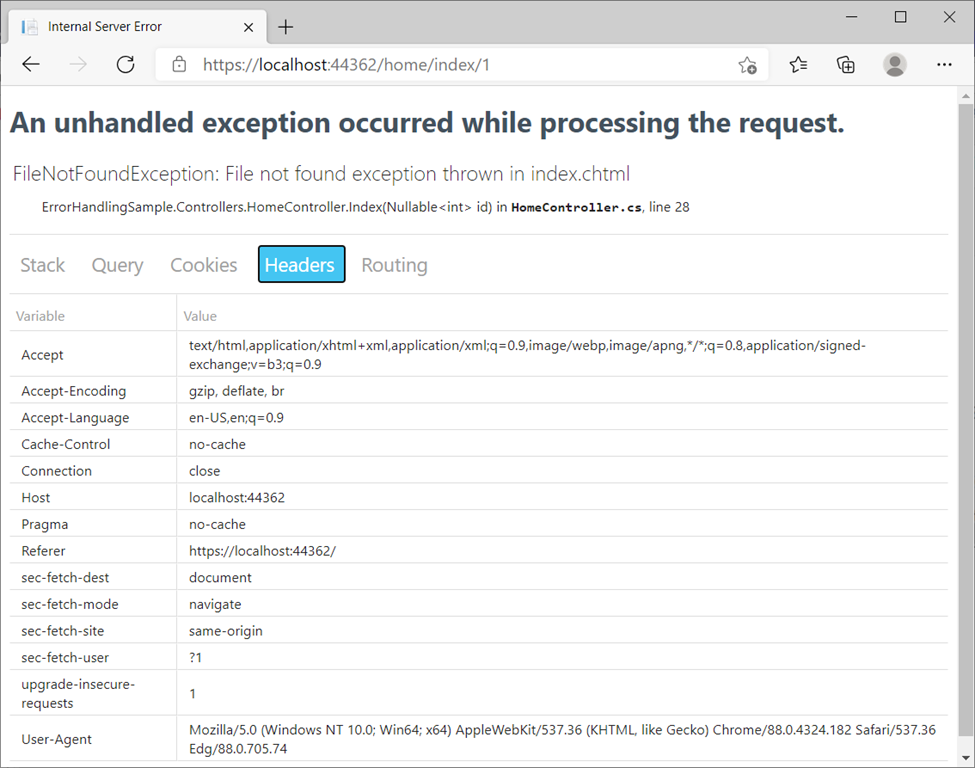
Click "Trigger an exception." you will get,

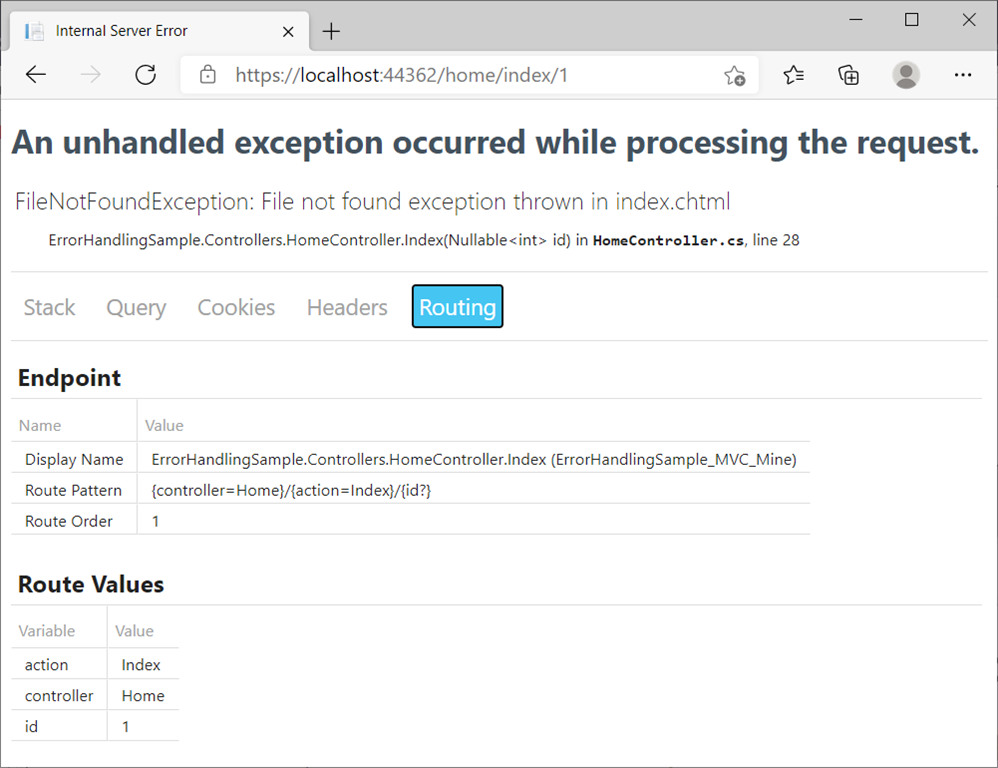


This is the Developer Exception Page that includes the following information about the exception and the request,

* Stack trace
* Query string parameters if any
* Cookies if any
* Headers
* Routing

For examples: Headers, and Routing





B: Exception Handling in Production Environment

ASP.NET Core configures app behavior based on the [runtime environment](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-5.0) that is determined in launchSettings.json file:

* [Development](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.environments.development) : The [launchSettings.json](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-5.0" \l "lsj" \t "_blank) file sets ASPNETCORE\_ENVIRONMENT to Development on the local machine.
* [Staging](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.environments.staging)
* [Production](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.environments.production) : The default if DOTNET\_ENVIRONMENT and ASPNETCORE\_ENVIRONMENT have not been set.

**Note**

The *launchSettings.json* file:

* Is only used on the local development machine.
* Is not deployed.
* contains profile settings.

Now, we switch environment from Development to Production:

1. {
2. "iisSettings": {
3. "windowsAuthentication": **false**,
4. "anonymousAuthentication": **true**,
5. "iisExpress": {
6. "applicationUrl": "http://localhost:50957",
7. "sslPort": 44362
8. }
9. },
10. "profiles": {
11. "IIS Express": {
12. "commandName": "IISExpress",
13. "launchBrowser": **true**,
14. "environmentVariables": {
15. //"ASPNETCORE\_ENVIRONMENT": "Development",
16. "ASPNETCORE\_ENVIRONMENT": "Production"
17. }
18. },
19. "ErrorHandlingSample": {
20. "commandName": "Project",
21. "dotnetRunMessages": "true",
22. "launchBrowser": **true**,
23. "applicationUrl": "https://localhost:5001;http://localhost:5000",
24. "environmentVariables": {
25. //"ASPNETCORE\_ENVIRONMENT": "Development",
26. "ASPNETCORE\_ENVIRONMENT": "Production"
27. }
28. }
29. }
30. }

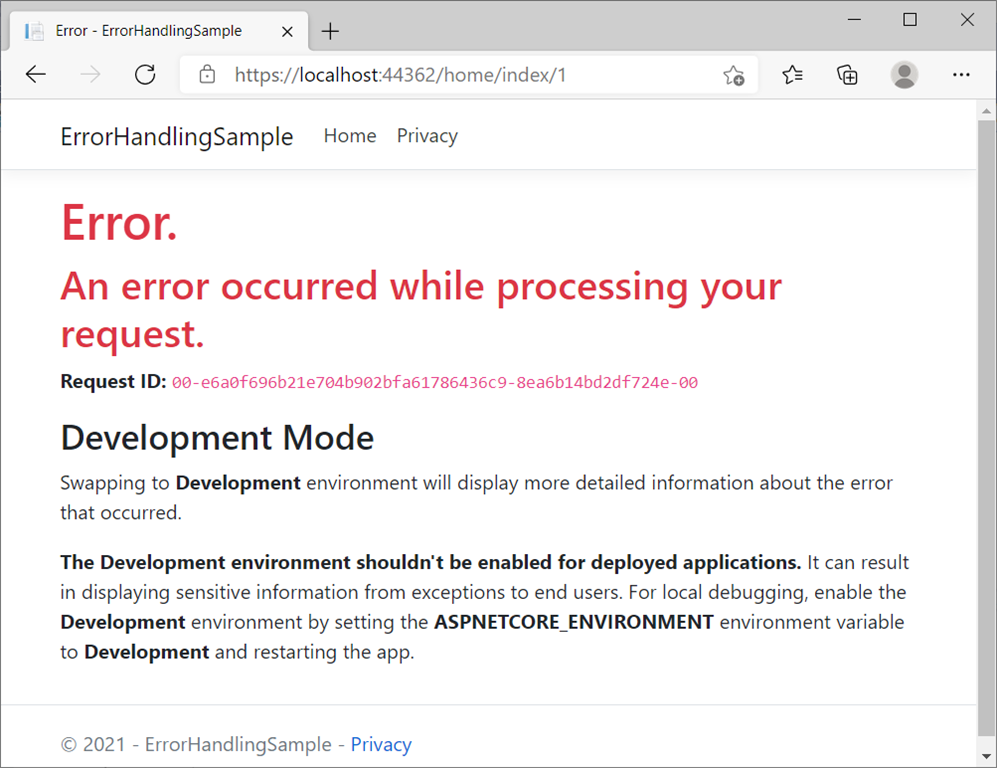
Approach 1: UseExceptionHandler

**1: Exception Handler Page**

For Production environment, startup file Configure method tells us: ASP.NET Core handles exception by calling UseExceptionHandler:

1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler("/Home/Error");
10. }
11. ......
12. }

Run the app, and Click **Trigger an exception** link in the home page, we got the Exception Handler Page, and by default *Home/Error.cshtml.cs* generated by the ASP.NET Core templates,



This exception handling middleware,

* Catches and logs exceptions.
* Re-executes the request in an alternate pipeline using the path indicated. The request isn't re-executed if the response has started. The template generated code re-executes the request using the /Home/Error path.

**2: Exception Handler Lambda**

An alternative to a [custom exception handler page](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-5.0#exception-handler-page) is to provide a lambda to [UseExceptionHandler](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.exceptionhandlerextensions.useexceptionhandler" \t "_blank). Using a lambda allows access to the error before returning the response.

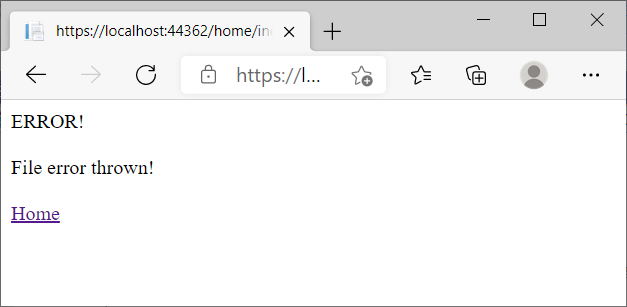
### Access the exception

Use [IExceptionHandlerPathFeature](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.diagnostics.iexceptionhandlerpathfeature) to access the exception and the original request path in an error handler

The following code uses a lambda for exception handling (startup file):

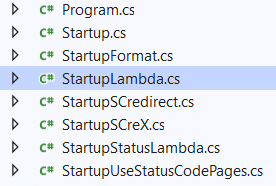
1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler(errorApp =>
10. {
11. errorApp.Run(async context =>
12. {
13. context.Response.StatusCode = 500;
14. context.Response.ContentType = "text/html";
16. await context.Response.WriteAsync("<html lang=\"en\"><body>\r\n");
17. await context.Response.WriteAsync("ERROR!<br><br>\r\n");
19. var exceptionHandlerPathFeature =
20. context.Features.Get<IExceptionHandlerPathFeature>();
22. **if** (exceptionHandlerPathFeature?.Error **is** FileNotFoundException)
23. {
24. await context.Response.WriteAsync(
25. "File error thrown!<br><br>\r\n");
26. }
28. await context.Response.WriteAsync(
29. "<a href=\"/\">Home</a><br>\r\n");
30. await context.Response.WriteAsync("</body></html>\r\n");
31. await context.Response.WriteAsync(**new** **string**(' ', 512));
32. });
33. });
34. app.UseHsts();
35. }
37. app.UseHttpsRedirection();
38. app.UseStaticFiles();
40. app.UseRouting();
42. app.UseAuthorization();
44. app.UseEndpoints(endpoints =>
45. {
46. endpoints.MapRazorPages();
47. });
48. }

We got the result,



**Note**

For convenience, we keep the original startup file as startup.cs file, and make a new startup file with a class name and file name as startupLambda, the highlighted one in the graph below,



and in Program.cs, comment out Startup class, replace it by startupLambda class, like this:

1. **public** **class** Program
2. {
3. **public** **static** **void** Main(**string**[] args)
4. {
5. CreateHostBuilder(args).Build().Run();
6. }
8. **public** **static** IHostBuilder CreateHostBuilder(**string**[] args) =>
9. Host.CreateDefaultBuilder(args)
10. .ConfigureWebHostDefaults(webBuilder =>
11. {
12. //webBuilder.UseStartup<Startup>();
13. webBuilder.UseStartup<StartupLambda>();
14. //webBuilder.UseStartup<StartupUseStatusCodePages>();
15. //webBuilder.UseStartup<StartupStatusLambda>();
16. //webBuilder.UseStartup<StartupFormat>();
17. //webBuilder.UseStartup<StartupSCredirect>();
18. //webBuilder.UseStartup<StartupSCreX>();
19. });
20. }

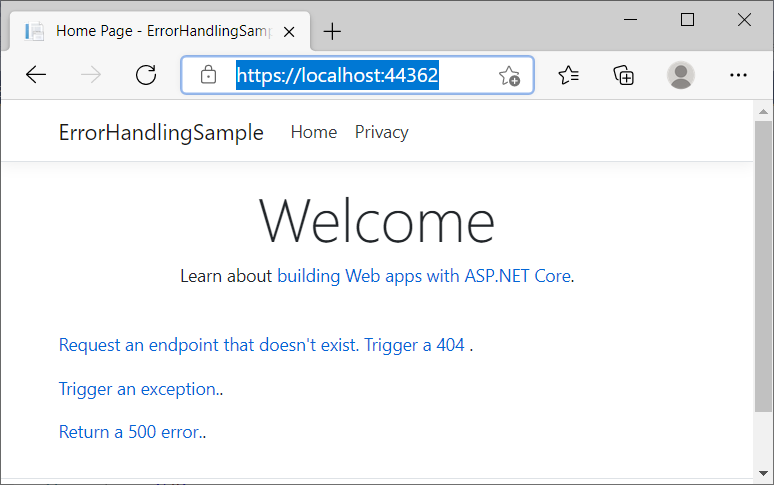
With similarity, we create several new startup classes as in the above graph, we will define them and use them in later discussions.

Approach 2: UseStatusCodePages

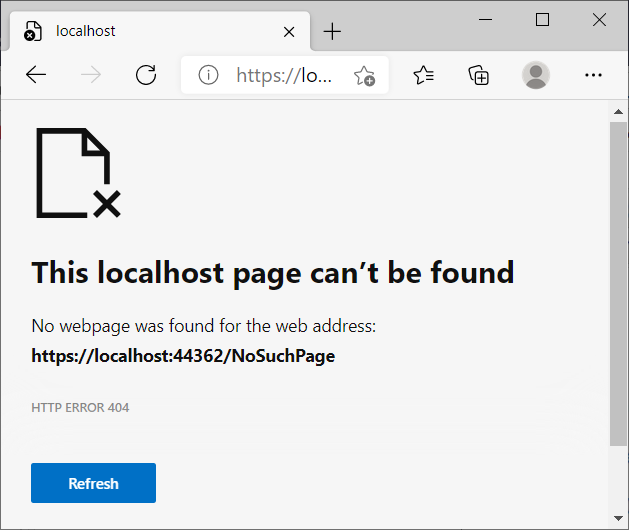
The two techniques discussed so far deal with the unhandled exceptions arising from code. However, that's not the only source of errors. Many times errors are generated due to internal server errors, non existent pages, web server authorization issues and so on. These errors are reflected by the HTTP status codes such as 500, 404 and 401.

By default, an ASP.NET Core app doesn't provide a status code page for HTTP error status codes, such as *404 - Not Found*. When the app encounters an HTTP 400-599 error status code that doesn't have a body, it returns the status code and an empty response body.

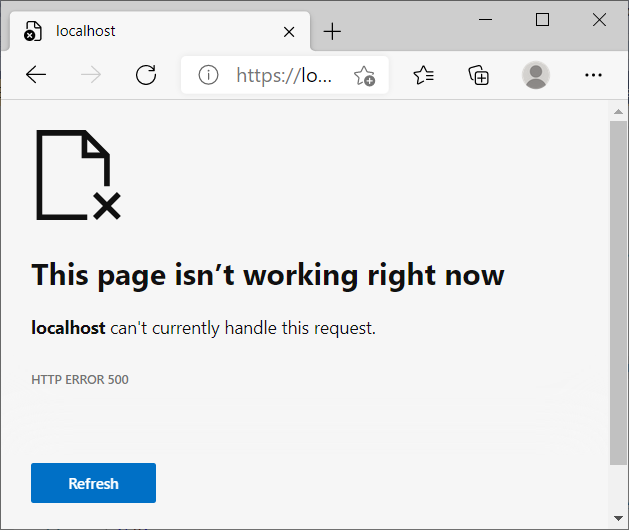
Click the link: Request an endpoint that doesn't exist. Trigger a 404 below,



We will get,



Whie clicking Run a 500 error,



To deal with such errors we can use UseStatusCodePages() method (status code pages middleware) to provide status code pages.

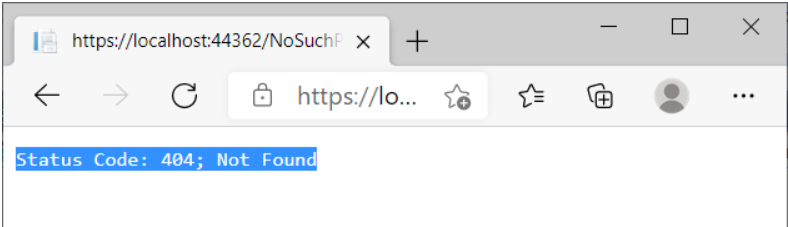
**1: Default UseStatusCodePages, or with format string, or with Lambda**

To enable default text-only handlers for common error status codes, call [UseStatusCodePages](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepages" \t "_blank) in the Startup.Configure method:

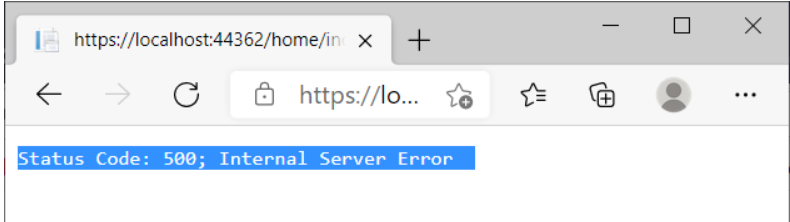
1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler("/Home/Error");
10. app.UseHsts();
11. }
13. app.UseStatusCodePages();
15. ......
16. }

We make this file (class) name as startupUseStatusCodePages, and  Remove the comments from webBuilder.UseStartup<StartupUseStatusCodePages>(); in Program.cs.

Run the app, the resullt will be:



for 404 error, and below for 500 error:



Again, we make startup file named as StartupFormat

1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler("/Home/Error");
10. app.UseHsts();
11. }
13. app.UseStatusCodePages(
14. "text/plain", "Status code page, status code: {0}");
15. ......
16. }

Remove the comments from webBuilder.UseStartup<StartupFormat>(); in Program.cs. Run the app, the resullt will be shown:

Status code page, status code: 404  
and

Status code page, status code: 500

br

br

The same, to make startup file named as StartupStatusLambda:

1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler("/Home/Error");
10. app.UseHsts();
11. }
13. app.UseStatusCodePages(async context =>
14. {
15. context.HttpContext.Response.ContentType = "text/plain";
17. await context.HttpContext.Response.WriteAsync(
18. "Status code lambda, status code: " +
19. context.HttpContext.Response.StatusCode);
20. });
21. ......
22. }

Remove the comments from webBuilder.UseStartup<StartupStatusLambda>(); in Program.cs. Run the app, the resullt will be shown:

Status code lambda, status code: 404

and

Status code lambda, status code: 500

**Note**

UseStatusCodePages isn't typically used in production because it returns a message that isn't useful to users.

**2: UseStatusCodePagesWithRedirects**

The [UseStatusCodePagesWithRedirects](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepageswithredirects" \t "_blank) extension method:

* Sends a status code to the client.
* Redirects the client to the error handling endpoint provided in the URL template. The error handling endpoint typically displays error information and returns HTTP 200.

Implementation

**Step 1: Set up Startup file**

Make startup file named as StartupSCredirect:

1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler("/Home/Error");
10. app.UseHsts();
11. }
13. app.UseStatusCodePagesWithRedirects("/Home/MyStatusCode?code={0}");
14. ......
15. }

Remove the comments from webBuilder.UseStartup<StartupSCredirect>(); in Program.cs.

**Step 2**:  Add an Action method in HomeController,

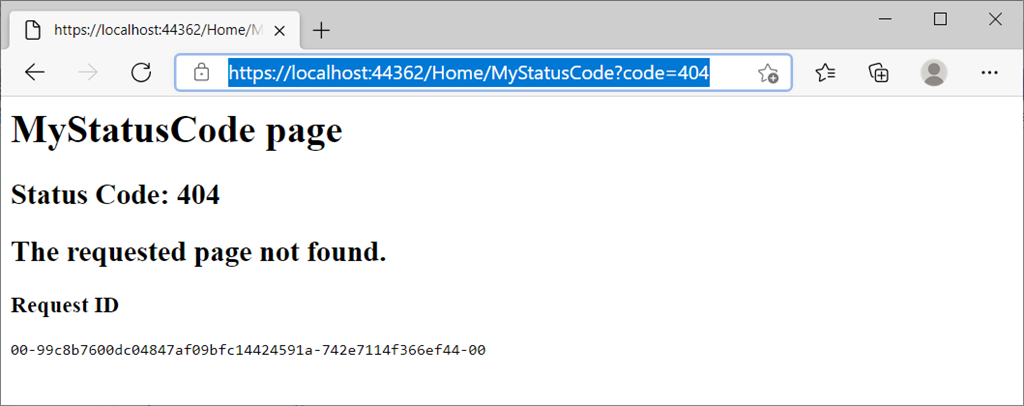
1. **public** IActionResult MyStatusCode(**int** code)
2. {
3. **if** (code == 404)
4. {
5. ViewBag.ErrorMessage = "The requested page not found.";
6. }
7. **else** **if** (code == 500)
8. {
9. ViewBag.ErrorMessage = "My custom 500 error message.";
10. }
11. **else**
12. {
13. ViewBag.ErrorMessage = "An error occurred while processing your request.";
14. }
16. ViewBag.RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;
17. ViewBag.ShowRequestId = !**string**.IsNullOrEmpty(ViewBag.RequestId);
18. ViewBag.ErrorStatusCode = code;
20. **return** View();
21. }

**Step 3**

Create a view for the Action: View/Home/MyStatusCode.cshtml

1. @{
2. Layout = **null**;  // clean up F12 tool network tab
3. }
5. @{ ViewData["Title"] = "Status Code @ViewBag.ErrorStatusCode"; }
6. <head>
7. <!-- prevent favicon.ico from being requested. -->
8. <link rel="icon" href="data:,">
9. </head>
11. <h1>MyStatusCode page</h1>
12. <h2 **class**="text-danger">Status Code: @ViewBag.ErrorStatusCode</h2>
13. <h2 **class**="text-danger"> @ViewBag.ErrorMessage</h2>
15. @**if** (ViewBag.ShowRequestId)
16. {
17. <h3>Request ID</h3>
18. <p>
19. <code>@ViewBag.RequestId</code>
20. </p>
21. }

Run the app, click either 400 or 500 errors, we got (for Error Code 400):



**Note**

The link is redirected to a new link that is endpoint provided.

**3: UseStatusCodePagesWithReExecute**

The [UseStatusCodePagesWithReExecute](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepageswithreexecute" \t "_blank) extension method:

* Returns the original status code to the client.
* Generates the response body by re-executing the request pipeline using an alternate path.

# StatusCodePagesExtensions.UseStatusCodePagesWithReExecute() : Adds a StatusCodePages middleware to the pipeline. Specifies that the response body should be generated by re-executing the request pipeline using an alternate path. This path may contain a '{0}' placeholder of the status code.

Implementation

**Step 1: Set up Startup file**

Make startup file named as StartupSCreX:

1. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3. **if** (env.IsDevelopment())
4. {
5. app.UseDeveloperExceptionPage();
6. }
7. **else**
8. {
9. app.UseExceptionHandler("/Home/Error");
10. app.UseHsts();
11. }
13. app.UseStatusCodePagesWithReExecute("/Home/MyStatusCode2", "?code={0}");
14. ......
15. }

Remove the comments from webBuilder.UseStartup<StartupSCreX>(); in Program.cs.

**Step 2**:  Add an Action method in HomeController,

1. **public** IActionResult MyStatusCode2(**int** code)
2. {
4. var statusCodeReExecuteFeature = HttpContext.Features.Get<
5. IStatusCodeReExecuteFeature>();
6. **if** (statusCodeReExecuteFeature != **null**)
7. {
8. ViewBag.OriginalURL =
9. statusCodeReExecuteFeature.OriginalPathBase
10. + statusCodeReExecuteFeature.OriginalPath
11. + statusCodeReExecuteFeature.OriginalQueryString;
12. }
14. ViewBag.RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;
15. ViewBag.ShowRequestId = !**string**.IsNullOrEmpty(ViewBag.RequestId);
16. ViewBag.ShowOriginalURL = !**string**.IsNullOrEmpty(ViewBag.OriginalURL);
17. ViewBag.ErrorStatusCode = code;
19. **return** View();
20. }

**Step 3**

Create a view for the Action: View/Home/MyStatusCode2.cshtml

@{

    Layout = **null**;  // clean up F12 tool network tab

}

@{ ViewData["Title"] = "Status Code @ViewBag.ErrorStatusCode"; }

<head>

    <!-- prevent favicon.ico from being requested. -->

    <link rel="icon" href="data:,">

</head>

<h1 **class**="text-danger">Status Code: @ViewBag.ErrorStatusCode</h1>

<h2 **class**="text-danger">An error occurred **while** processing your request.</h2>

@**if** (ViewBag.ShowRequestId)

{

<h3>Request ID</h3>

                <p>

                    <code>@ViewBag.RequestId</code>

                </p>}

@**if** (ViewBag.ShowOriginalURL)

{

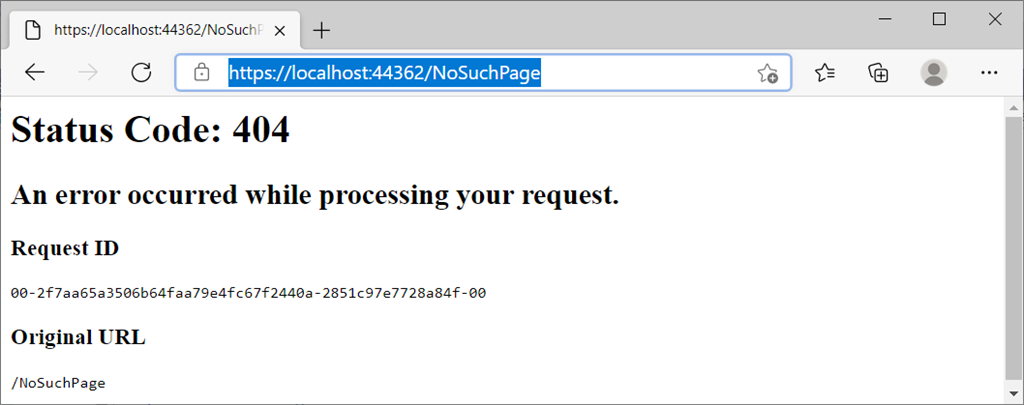
<h3>Original URL</h3>

                <p>

                    <code>@ViewBag.OriginalURL</code>

                </p>}

Run the app, click either 400 or 500 errors, we got (for Error Code 400):



**Note**

The link is kept the same with original one.

Approach 3: Exception Filter

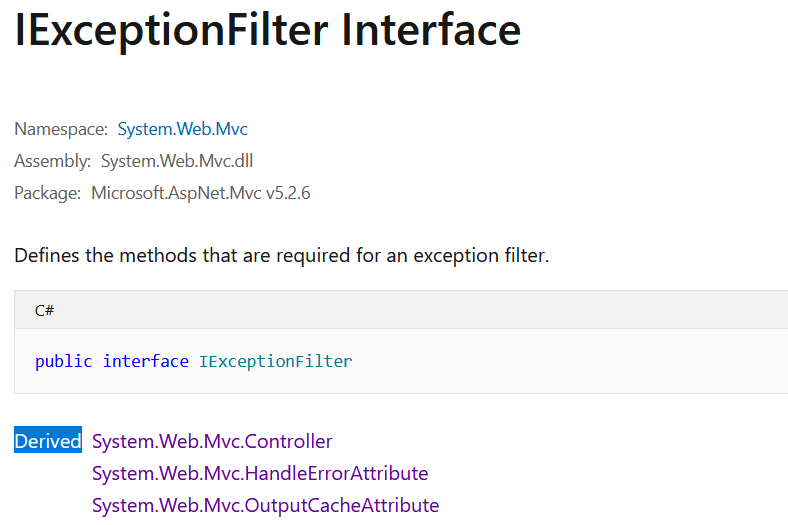
**General Discussion**

In MVC apps, exception filters can be configured globally or on a per-controller or per-action basis. In Razor Pages apps, they can be configured globally or per page model. These filters handle any unhandled exceptions that occur during the execution of a controller action or another filter.

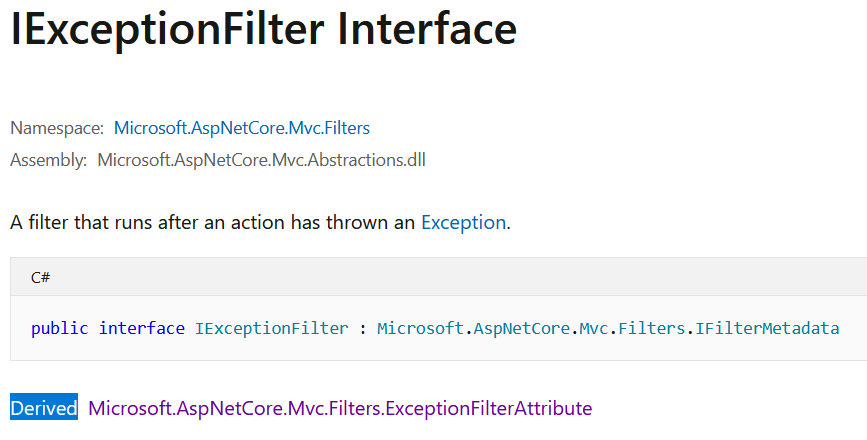
Exception filters are useful for trapping exceptions that occur within MVC actions, but they're not as flexible as the built-in exception handling middleware, UseExceptionHandler. [Microsoft](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-5.0#startup-exception-handling) recommend using UseExceptionHandler, unless you need to perform error handling differently based on which MVC action is chosen.

**Difference from ASP.NET MVC**

1. In ASP.NET MVC, Exception Filter is the major approach for exception handling, while for ASP.NET Core MVC, as Microsoft suggested, the built-in exception hadling middleware, UseExceptionHandler, is more flexible and suitable.
2. IExceptionFilter Interface for ASP.NET is derived by [System.Web.Mvc.HandleErrorAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.handleerrorattribute?view=aspnet-mvc-5.2" \t "_blank) and System.Web.Mvc.Controller, therefore, we can either overriding OnException method from a class derived from HandleErrorAttribute class, or directly overriding OnException method from a controller. However, IExceptionFilter Interface for ASP.NET Core is only derived by [Microsoft.AspNetCore.Mvc.Filters.ExceptionFilterAttribute](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.filters.exceptionfilterattribute?view=aspnetcore-5.0" \t "_blank), not by Controller any more. So, we have to implemente IExceptionFilter interface directly or from ExceptionFilterAttribute class, but not from Controller directly any more.



**ASP.NET**



ASP.NET Core

Exception filters

* Implement [IExceptionFilter](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.filters.iexceptionfilter" \t "_blank) or [IAsyncExceptionFilter](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.filters.iasyncexceptionfilter" \t "_blank).
* Can be used to implement common error handling policies.

The following sample exception filter uses a custom error view to display details about exceptions that occur when the app is in development:

Implemantation

**Step 1**

Create an Custom Exception Filter: CustomExceptionFilter

1. **using** Microsoft.AspNetCore.Mvc;
2. **using** Microsoft.AspNetCore.Mvc.Filters;
3. **using** Microsoft.AspNetCore.Mvc.ModelBinding;
4. **using** Microsoft.AspNetCore.Mvc.ViewFeatures;
6. **namespace** ErrorHandlingSample.Filters
7. {
8. **public** **class** CustomExceptionFilter : IExceptionFilter
9. {
10. **private** **readonly** IModelMetadataProvider \_modelMetadataProvider;
12. **public** CustomExceptionFilter(IModelMetadataProvider modelMetadataProvider)
13. {
14. \_modelMetadataProvider = modelMetadataProvider;
15. }
17. **public** **void** OnException(ExceptionContext context)
18. {
19. var result = **new** ViewResult { ViewName = "CustomError" };
20. result.ViewData = **new** ViewDataDictionary(\_modelMetadataProvider, context.ModelState);
21. result.ViewData.Add("Exception", context.Exception);
23. // Here we can pass additional detailed data via ViewData
24. context.ExceptionHandled = **true**; // mark exception as handled
25. context.Result = result;
26. }
27. }
28. }

**Step 2**

Create a CustomError view: View/Shared/CustomError.cshtml

1. @{
2. ViewData["Title"] = "CustomError";
3. var exception = ViewData["Exception"] **as** Exception;
4. }
6. <h1>An Error has Occurred</h1>
8. <p>@exception.Message</p>

**Step 3**

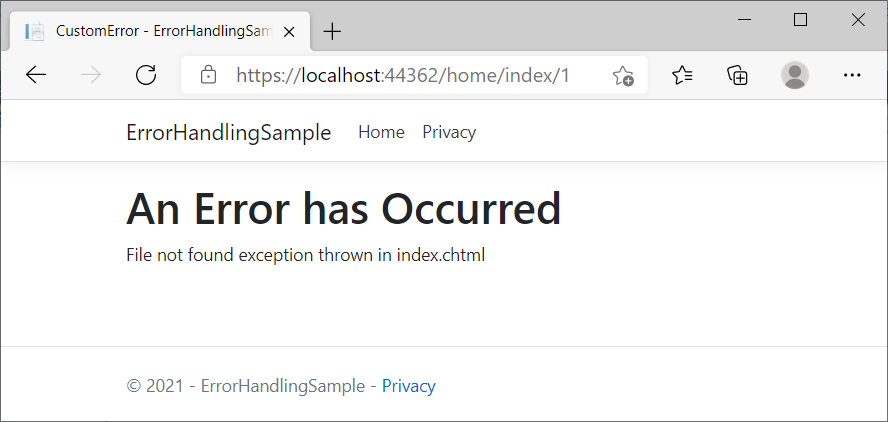
Register in either locally in Controller level or Action level, e.g.

1. [TypeFilter(**typeof**(CustomAsyncExceptionFilter))]
2. **public** IActionResult Failing()
3. {
4. **throw** **new** Exception("Testing custom exception filter.");
5. }

or global level in startup.ConfigureService:

1. **public** **void** ConfigureServices(IServiceCollection services)
2. {
3. services.AddControllersWithViews();
5. services.AddControllersWithViews(config => config.Filters.Add(**typeof**(CustomExceptionFilter)));
6. }

Run the app, and Test it: Click Trigger an exception (you must either register the Exception filter locally in Action or Controller or Globally):



Summary

* A: Exception Handling in Development Environment for ASP.NET Core MVC
  + UseDeveloperExceptionPage
* B: Exception Handling in Production Environment for ASP.NET Core MVC
  + Approach 1: **UseExceptionHandler**
    - 1: Exception Handler Page
    - 2: Exception Handler Lambda
  + Approach 2: **UseStatusCodePages**
    - 1: UseStatusCodePages, and with format string, and with Lambda
    - 2: UseStatusCodePagesWithRedirects
    - 3: UseStatusCodePagesWithReExecute
  + Approach 3: **Exception Filter**
    - Local
    - Global