4.Exception Handling in .net core

Exception handling is one of the most important features of any application. .NET Core includes a middleware that makes exception handling easy.

By default, .NET Core returns a simple status code for any exception that occurs in an application.

❖ **UseDeveloperExceptionPage**

The major purpose of UseDeveloperExceptionPage() is to help the user to inspect exception details during the development phase.

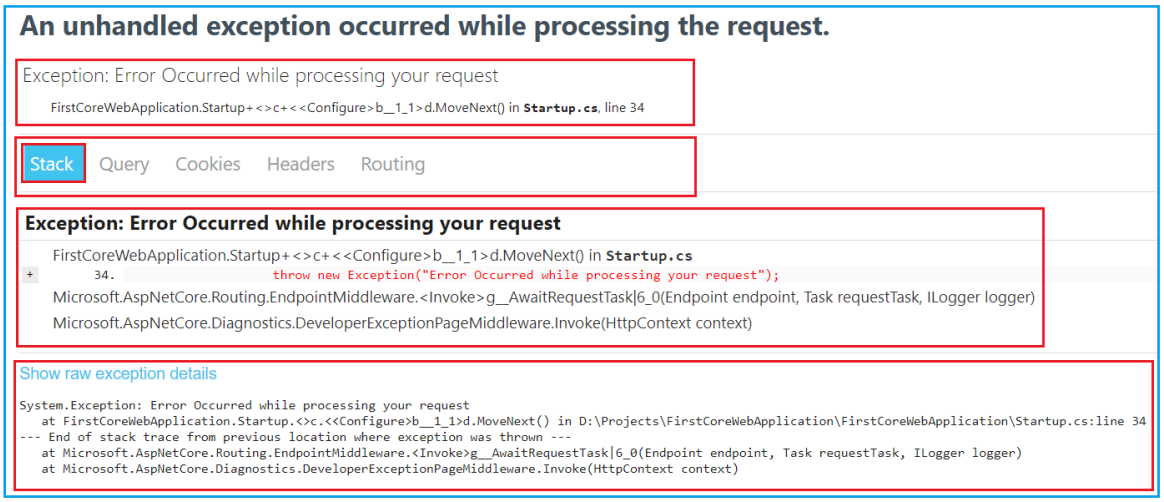
- **Purpose of UseDeveloperExceptionPage**

To capture Synchronous and Asynchronous SystemException instance from the pipeline & generate HTML error response.
It returns a reference to the application after the operation is completed.

- We use the UseDeveloperException() extension method to render the exception during the development mode
- This method adds middleware into the request pipeline which displays a developer-friendly exception detail page. This helps developers in tracing errors that occur during the development phase

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
if (env.IsDevelopment())
{
        app.UseDeveloperExceptionPage();
}
app.UseRouting();
app.UseEndpoints(endpoints =>
{
        endpoints.MapGet("/", async context =>
        {
        throw new Exception("Error Occurred while processing your request");
        await context.Response.WriteAsync("Hello World!");
        });
});
}
```

With the above change in place, now run the application and it should display the following page with the detailed information about the unhandled exception.



As you can see in the above image, the Developer Exception Page contains five tabs such as Stack, Queue, Cookies, Headers, and Routing.

- **Stack**: The Stack tab gives the information of stack trace which indicates where exactly the exception occurred, the file name, and the line number that causes the exception.
- **Query**: The Query tab gives information about the query strings.
- **Cookies**: The Cookies tab displays the information about the cookies set by the request.
- **Header**: The Header tab gives information about the headers which is sent by the client when makes the request.
- **Route**: The Route tab gives information about the Route Pattern and Route HTTP Verb type of the method, etc.

Now if you verify the Query tab and Cookies tab, then you will not see any information as you are not passing any query string value in the URL or you are not setting the cookies in the request. In our upcoming articles, we will discuss the Query string and Cookies in detail.

❖ UseExceptionHandler

To configure a custom error handling page for the Production environment, call UseExceptionHandler. This exception handling middleware:

- Catches and logs unhandled 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 /Error path.

- **Access the exception**

Use IExceptionHandlerPathFeature to access the exception and the original request path in an error handler. The following example uses IExceptionHandlerPathFeature to get more information about the exception that was thrown:

```
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
[IgnoreAntiforgeryToken]
public class ErrorModel : PageModel
{
public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

public string? ExceptionMessage { get; set; }

public void OnGet()
{
        RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;

        var exceptionHandlerPathFeature =
        HttpContext.Features.Get<IExceptionHandlerPathFeature>();

        if (exceptionHandlerPathFeature?.Error is FileNotFoundException)
        {
        ExceptionMessage = "The file was not found.";
        }

        if (exceptionHandlerPathFeature?.Path == "/")
        {
        ExceptionMessage ??= string.Empty;
        ExceptionMessage += " Page: Home.";
        }
}
}
```