Exception Handling Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Built-In Global Exception Handling

Open the **CustomerRouter.cs** file and in the Get() method, add the following line of code just after the line **List<Customer> list;**

```
// Intentionally Cause an Exception
throw new ApplicationException("ERROR!");
```

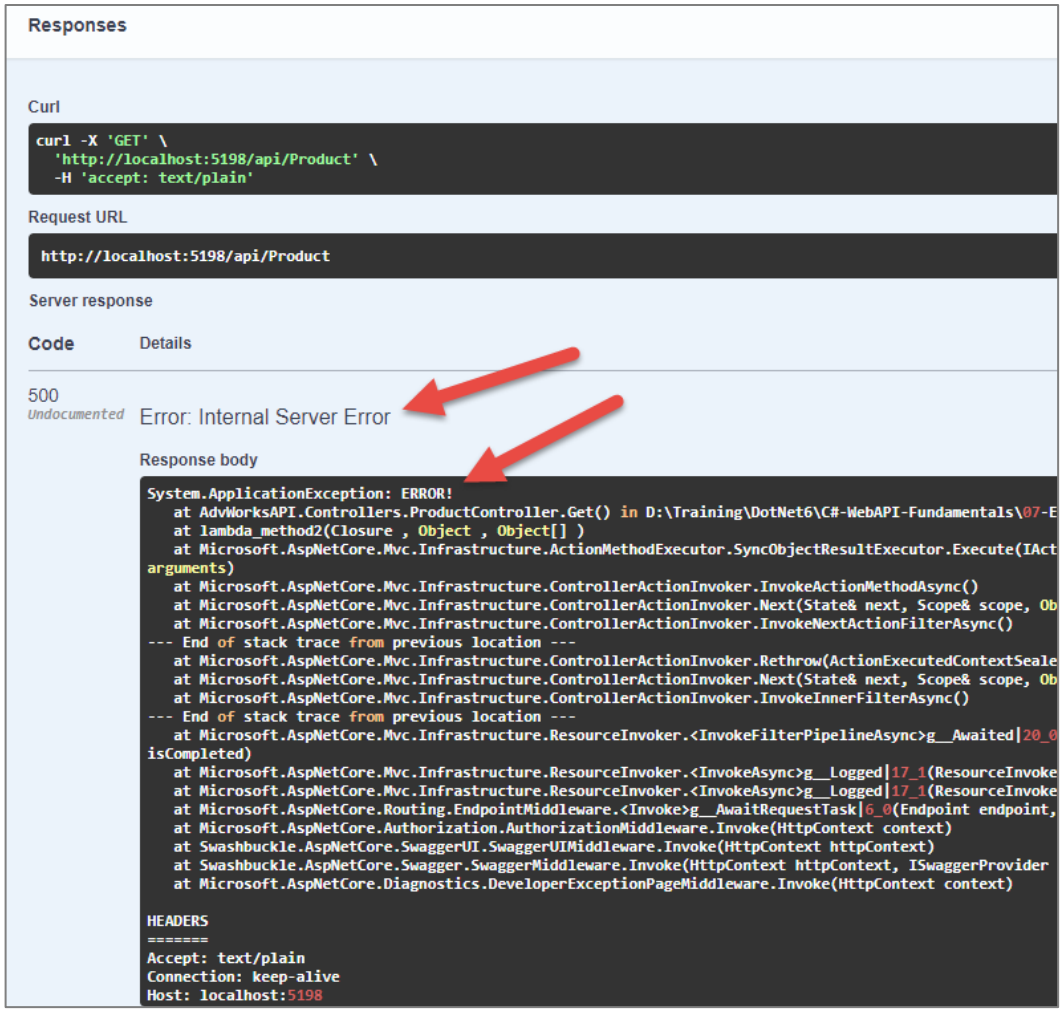
Try it Out

Run the application.

Click on the **GET /api/Customer** button.

You may need to go back to Visual Studio and click **Continue**.

View the exception that is returned (500 Error: Internal Server Error) similar to that in the screen shot below.



Lab 2: Add Error Handling Router

Right mouse-click on the RouterClasses folder and add a new class named **ErrorRouter**. Replace the entire contents of this new file with the following code.

```
using Microsoft.AspNetCore.Diagnostics;

namespace AdvWorksAPI.RouterClasses;

public class ErrorRouter : RouterBase
{
  /// <summary>
  /// Add routes
  /// </summary>
  /// <param name="app">A WebApplication object</param>
  public override void AddRoutes(WebApplication app)
  {
    app.Map("/ProductionError", (HttpContext context) =>
ProductionErrorHandler(context));
  }

  protected virtual IResult
ProductionErrorHandler(HttpContext context)
  {
    string msg = "Unknown Exception";

    var features =
context.Features.Get<IExceptionHandlerFeature>();

    if (features != null) {
      msg = features.Error.Message;
    }

    return Results.Problem(msg);
  }
}
```

Open the **Program.cs** file and register the ErrorRouter into DI

```
builder.Services.AddScoped<RouterBase, ErrorRouter>();
```

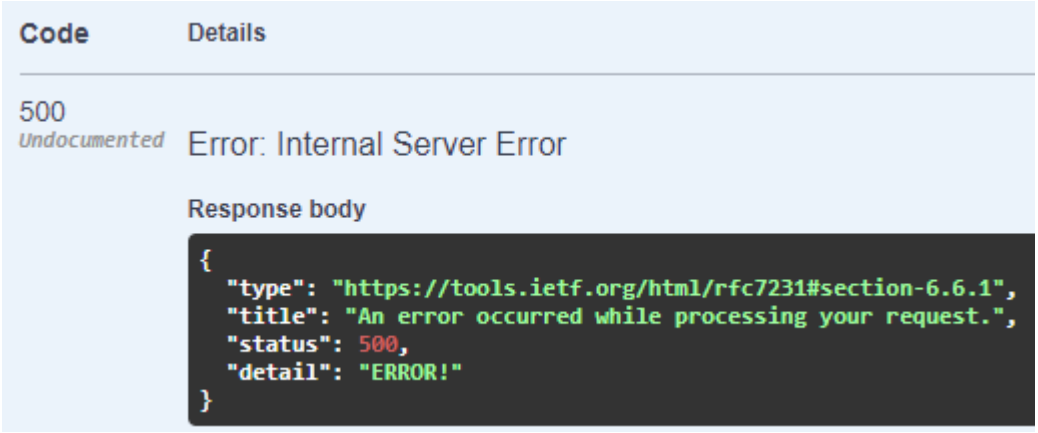
Add the following line of code just before the **using** block with the app.Services.CreateScope() method.

```
// Enable Exception Handling Middleware
app.UseExceptionHandler("/ProductionError");
```

Try it Out

Run the application and click on the **GET /api/Customer** button

The JSON object is what you get when you use the Result.Problem() method.



Lab 3: Development and Production Exception Handling

Open the **Program.cs** file and replace the code from the last demo to the following lines of code:

```
// Enable Exception Handling Middleware
if (app.Environment.IsDevelopment()) {
  app.UseExceptionHandler("/DevelopmentError");
}
else {
  app.UseExceptionHandler("/ProductionError");
}
```

Open the **ErrorRouter.cs** file and add a new method named **DevelopmentErrorHandler**().

```
protected virtual IResult
DevelopmentErrorHandler(HttpContext context)
{
  string msg = "Unknown Exception";

  var features =
context.Features.Get<IExceptionHandlerFeature>();

  if (features != null) {
    msg = "Message: " + features.Error.Message;
    msg += Environment.NewLine + "Source: " +
features.Error.Source;
    msg += Environment.NewLine +
features.Error.StackTrace;
  }

  return Results.Problem(msg);
}
```

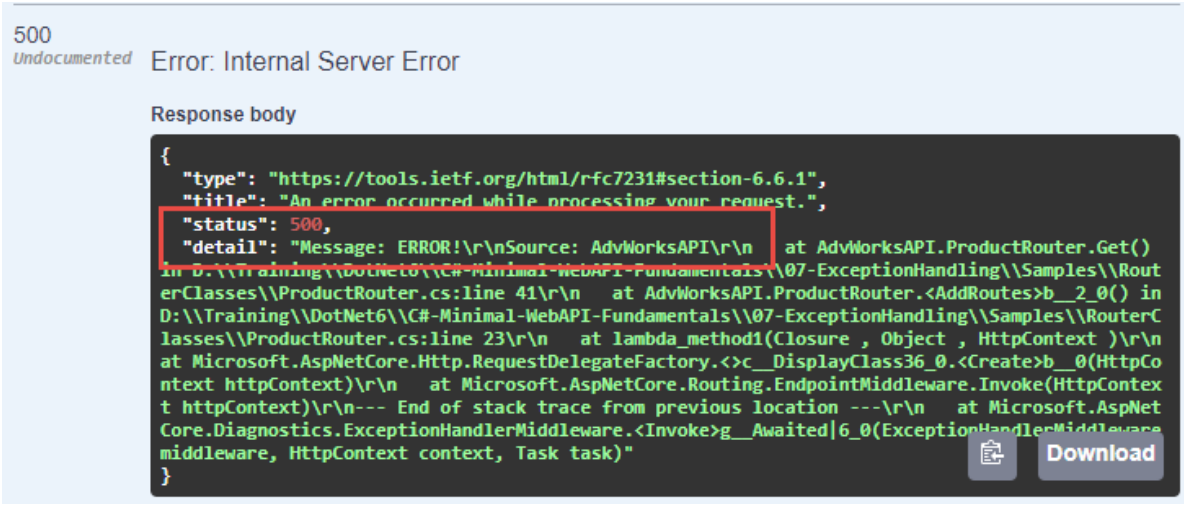
Add a new app.MapGet() to the AddRoutes() method.

```
app.Map("/DevelopmentError", (HttpContext context) =>
DevelopmentErrorHandler(context));
```

Try it Out

Run the application and click on the **GET /api/Customer** button

You should now see the message and the source and the stack trace as shown below.



Lab 4: Add Production Profile

Open the **\Properties\launchSettings.json** file.

Copy the JSON object "AdvWorksAPI" to a new JSON object named "**AdvWorksAPI Production"**

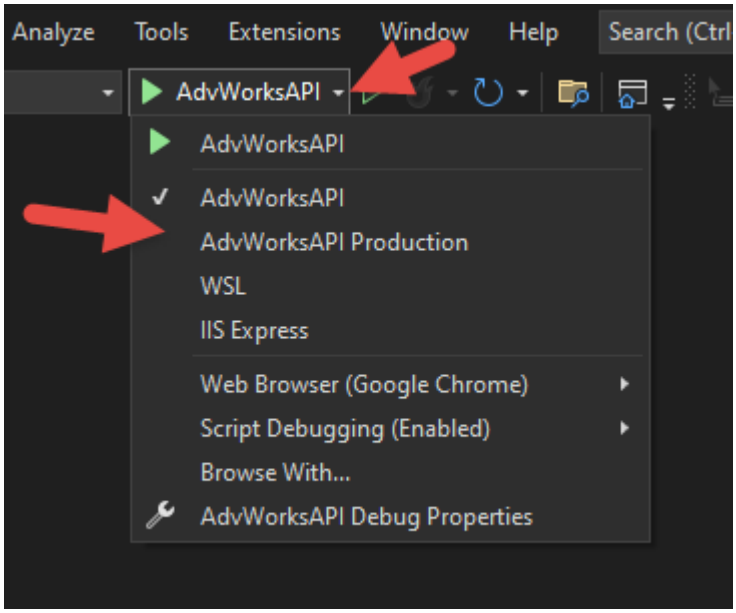
Change the areas in the copied JSON object with the values in **bold** below.

```
"AdvWorksAPI Production": {
  "commandName": "Project",
  "dotnetRunMessages": true,
  "launchBrowser": true,
  "launchUrl": "swagger",
  "applicationUrl": "http://localhost:5198",
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Production"
  }
},
```

Save the **launchSettings.json** file

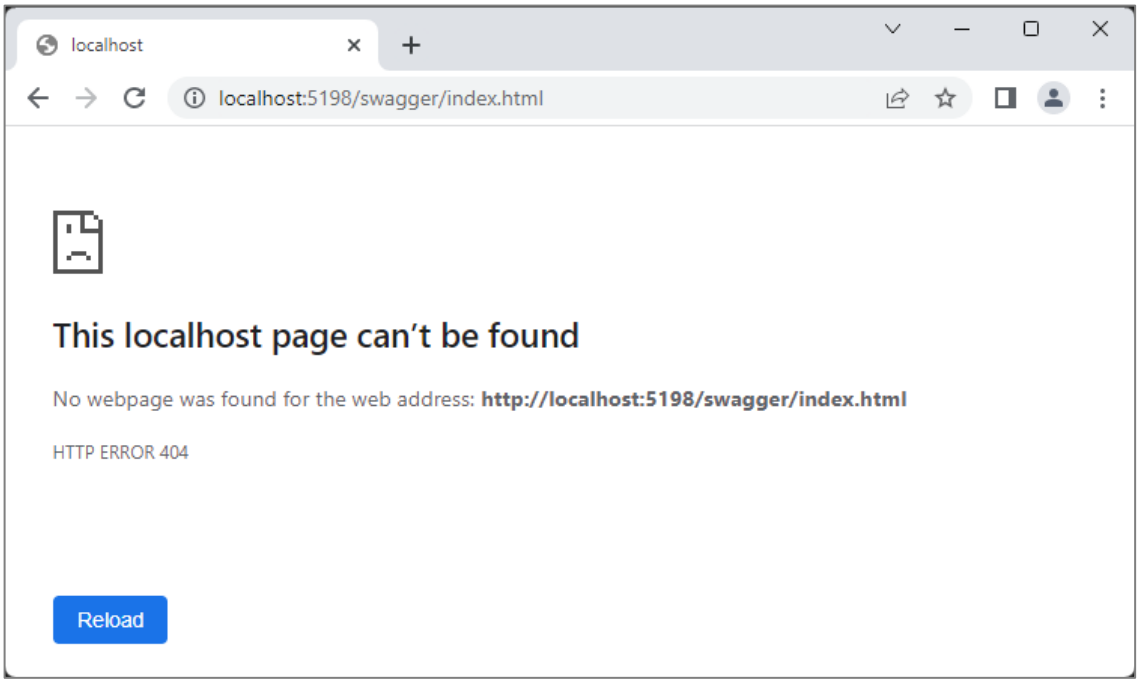
Try it Out

Click on the down arrow next to "AdvWorksAPI" on the VS command bar.



Select the "**AdvWorksAPI Production**" profile.

Run the application and you will get a 404 error.



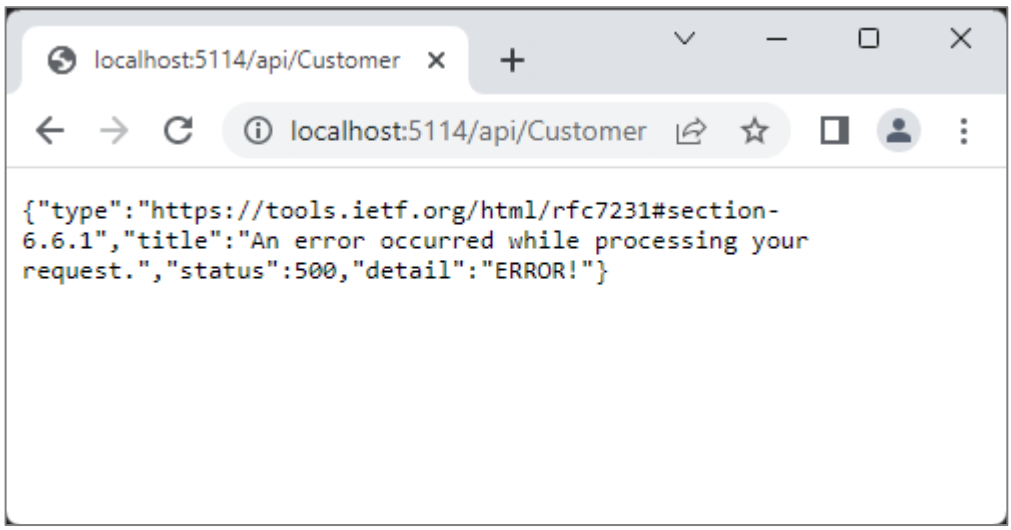
NOTE: Swagger does not appear when running in Production mode.

Type the following into the browser URL line (replacing your **PORT** number for the **nnnn**).

```
http://localhost:nnnn/api/Customer
```

Click **Continue** in Visual Studio when the exception occurs.

Now you just have the error message returned in the browser as shown below.



Lab 5: Handle Status Codes

Open the **ErrorRouter.cs** file

Add a new method.

```
protected virtual IResult StatusCode(int code,
HttpContext context)
{
  string msg = string.Empty;

  // Get some path information
  var feature =
context.Features.Get<IStatusCodeReExecuteFeature>();
  if (feature != null) {
    msg = feature.OriginalPathBase
        + feature.OriginalPath
        + feature.OriginalQueryString;
  }

  switch (code) {
    case 404:
      msg = $"API Route Was Not Found: '{msg}'";
      break;
    default:
      msg = $"Status Code Not Handled: '{code}'";
      break;
  }

  return Results.Problem(msg, statusCode: code);
}
```

Add a new app.Map()

```
app.Map("/StatusCode/{code:int}", (int code, HttpContext
context) => StatusCode(code, context));
```

Open **Program.cs** and just below the app.UseExceptionHandler() add

```
// Handle Other Status Codes
app.UseStatusCodePagesWithReExecute("/StatusCode/{0}");
```

Try it Out

While still in production mode, run the app.

You should now see the 404 status returned because swagger is not found.

Lab 6: Log Exceptions and Informational Messages into Different Files

Open the **Program.cs** file and modify the configuration of SeriLog so you have two files: one for informational messages and higher and the other for exceptions.

```
// Configure logging to Console & File using Serilog
builder.Host.UseSerilog((ctx, lc) =>
{
  // Log to Console
  lc.WriteTo.Console();
  // Log to Rolling File
  lc.WriteTo.File("Logs/InfoLog-.txt",
    rollingInterval: RollingInterval.Day,
    restrictedToMinimumLevel:
LogEventLevel.Information);
  // Log Exceptions to a Rolling File
  lc.WriteTo.File("Logs/ErrorLog-.txt",
    rollingInterval: RollingInterval.Day,
    restrictedToMinimumLevel: LogEventLevel.Error);
});
```

Try it Out

Delete any log files under the **Logs** folder.

Switch back to the **AdvWorksAPI** profile.

Run the application and click on the **GET /api/Customer** button.

Stop the application.

View the **Logs** folder and you should see two different log files.

NOTE:	You still get exceptions in the **InfoLog.txt** file because you can only set the minimum level. Look up Serilog.Filters.Expressions and how to configure which log levels go into which files.

Lab 7: Log Exceptions in Catch Block

In this lab you add a try…catch block and log your own custom messages.

Open the **CustomerRouter.cs** file and add a new field

```
private readonly ILogger<CustomerRouter> _Logger;
```

Modify the constructor

```
public CustomerRouter(IRepository<Customer> _repo,
  ILogger<CustomerRouter> logger)
{
  UrlFragment = "api/Customer";
  TagName = "Customer";
  _Repo = _repo;
  _Logger = logger;
}
```

In the AddRoutes() method add .Produces(500) on the MapGet() for the Get() method.

```
app.MapGet($"/{UrlFragment}", () => Get())
    .WithTags(TagName)
    .Produces(200)
    .Produces<List<Customer>>()
    .Produces(404)
    .Produces(500);
```

Modify the Get() method

```
protected virtual IResult Get()
{
  IResult ret;
  List<Customer> list;
  string msg = "No Customers Found.";

  try {
    // Intentionally Cause an Exception
    throw new ApplicationException("ERROR!");

    list = _Repo.Get();

    //list.Clear();
    if (list == null || list.Count == 0) {
      ret = Results.NotFound(msg);
    }
    else {
      ret = Results.Ok(list);
    }
  }
  catch (Exception ex) {
    msg = "Error in CustomerRouter.Get()";
    msg += $"{Environment.NewLine}Message:
{ex.Message}";
    msg += $"{Environment.NewLine}Source: {ex.Source}";

    // Log error for the developer
    _Logger.LogError(ex, "{msg}", msg);

    // Return generic message for the user
    ret = Results.Problem("Error in Customer API. Please
Contact the System Administrator.");
  }

  return ret;
}
```

Try it Out

Delete any log files in the **Logs** folder.

Run the application and click on the **GET /api/Customer** button.

See the error displayed.

Check the **ErrorLog-nnnn.txt** file.