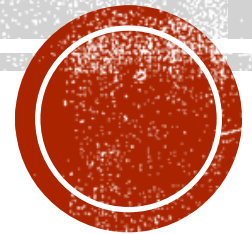


HANDLING REQUESTS WITH THE MIDDLEWARE PIPELINE

Kamal Beydoun

Lebanese University – Faculty of Sciences I

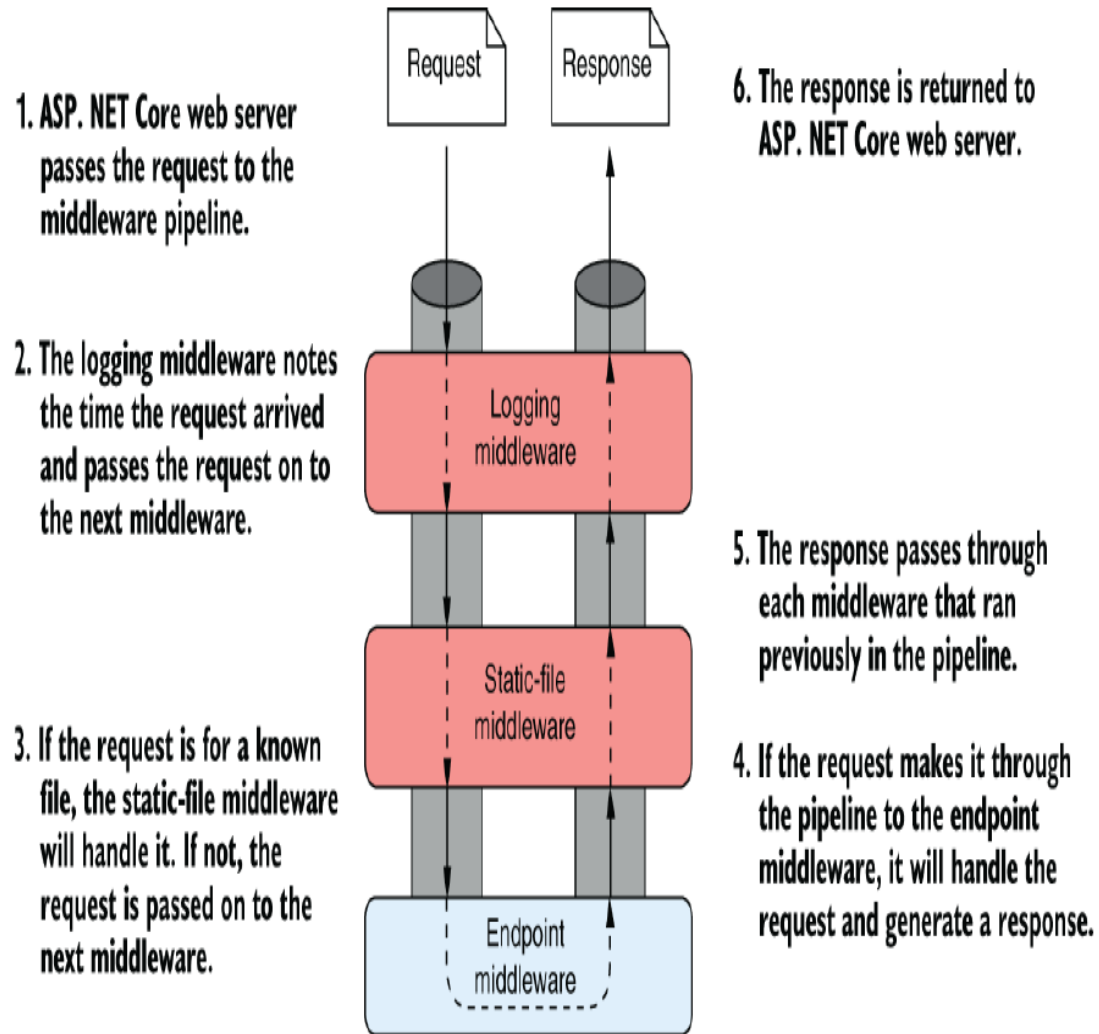
Kamal.beydoun@ul.edu.lb



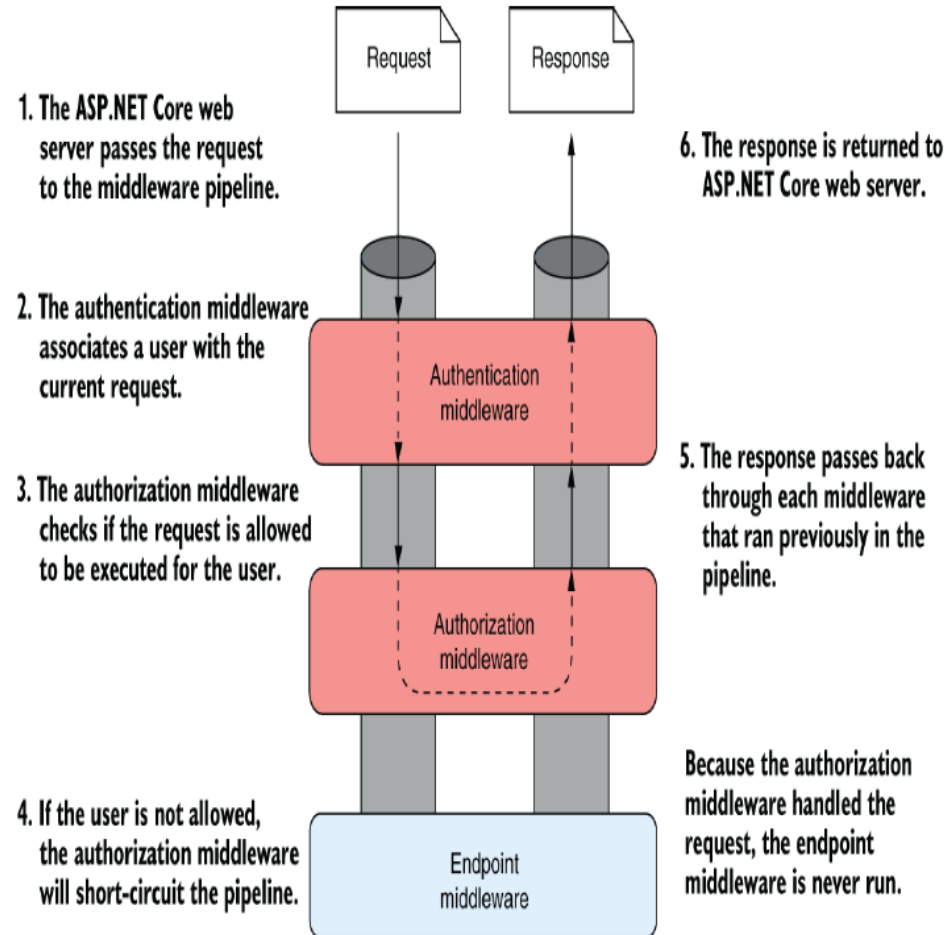
DEFINING MIDDLEWARE

- **ASP.NET Core Middleware:**
 - Comprises C# classes or functions managing HTTP requests or responses.
 - Chained together, creating a pipeline.
 - **Output** of one middleware serves as the **input** for the next in the sequence.
- **ASP.NET Core Middleware can:**
 - Handle an incoming HTTP request by generating an HTTP response
 - Process an incoming HTTP request, modify it, and pass it on to another piece of middleware
 - Process an outgoing HTTP response, modify it, and pass it on to another piece of middleware or to the ASP.NET Core web server

DEFINING MIDDLEWARE



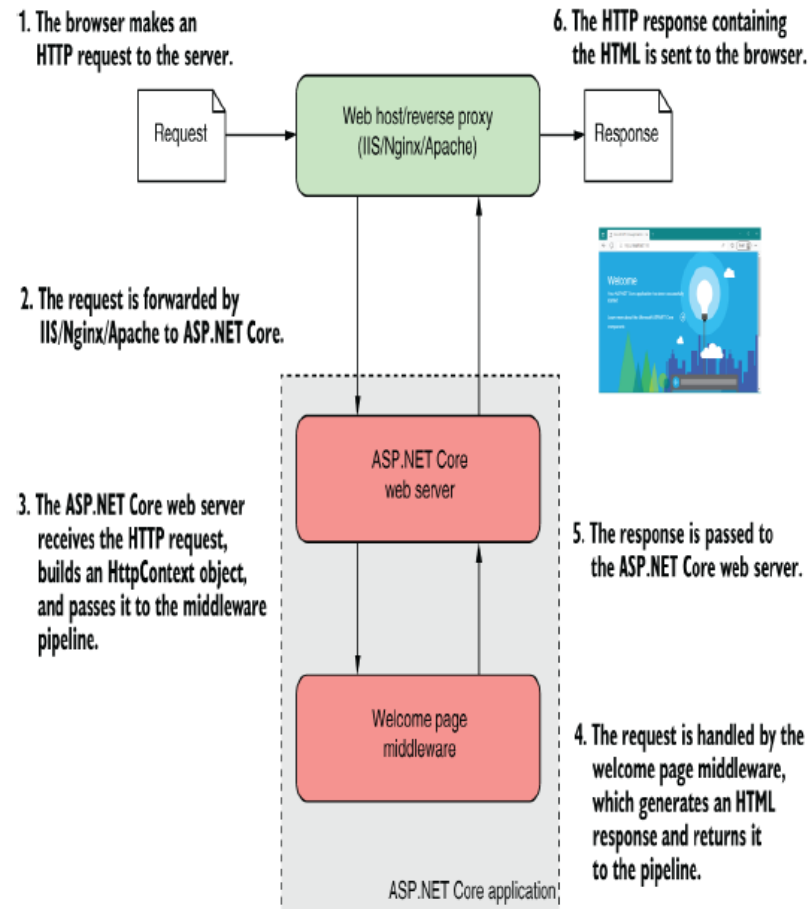
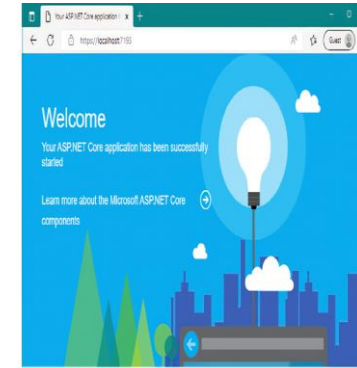
DEFINING MIDDLEWARE



HTTPCONTEXT OBJECT

- **Construction by Web Server:**
 - For each **request**, ASP.NET Core web server builds an **HttpContext**.
 - Serves as a storage box for individual requests.
- **Contents of HttpContext:**
 - Properties of the request, request-specific services, loaded data, and errors.
- **Middleware Access to HttpContext:**
 - All middleware has access to the HttpContext for a request.
 - Utilizes this object to examine user credentials, identify requested pages, and retrieve posted data.
- **Processing and Updating HttpContext:**
 - Application processes the request and updates the HttpContext with an appropriate response.
 - Returned through the middleware pipeline to the web server.
- **Response Handling:**
 - ASP.NET Core web server converts the updated HttpContext to a raw HTTP response.
 - Sent back to the reverse proxy, which forwards it to the user's browser.

COMBINING MIDDLEWARE IN A PIPELINE — A HOLDING PAGE



```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);  
WebApplication app = builder.Build();  
  
app.UseWelcomePage();  
  
app.Run();
```

1
1
2
3

COMBINING MIDDLEWARE IN A PIPELINE - HANDLING STATIC FILES

1. An HTTP request is made for the file moon.jpg.

7. The HTTP response containing the file is sent to the browser.

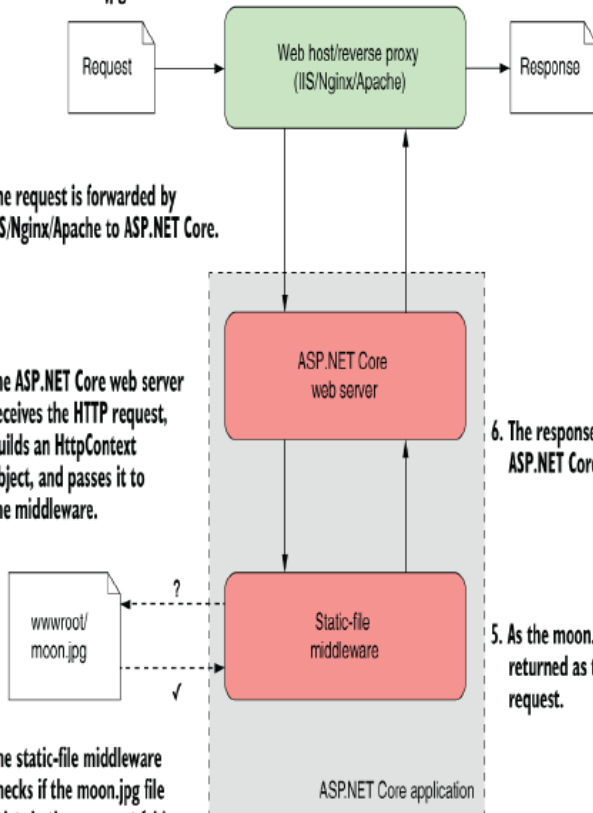
2. The request is forwarded by IIS/Nginx/Apache to ASP.NET Core.

3. The ASP.NET Core web server receives the HTTP request, builds an HttpContext object, and passes it to the middleware.

4. The static-file middleware checks if the moon.jpg file exists in the wwwroot folder, and if so, retrieves it.

6. The response is passed to the ASP.NET Core web server.

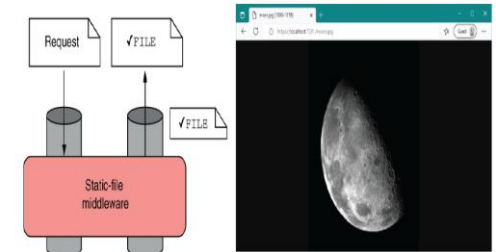
5. As the moon.jpg file exists, it is returned as the response to the request.



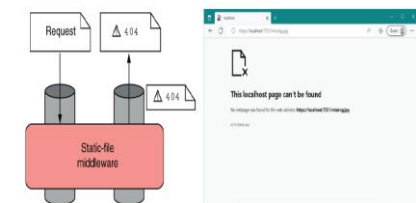
```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
WebApplication app = builder.Build();

app.UseStaticFiles(); ❶

app.Run();
```



1. The static-file middleware handles the request by returning the file requested.
2. The file stream is sent back through the middleware pipeline and out to the browser.
3. The browser displays the file returned in the response.



1. The static-file middleware handles the request by trying to return the requested file, but as it doesn't exist, it returns a raw 404 response.
2. The 404 HTTP error code is sent back through the middleware pipeline to the user.
3. The browser displays its default "File Not Found" error page.

HTTP RESPONSE STATUS CODES

- Every HTTP response contains a *status code* and, optionally, a *reason phrase* describing the status code.
 - **1xx—Information.** This code is not often used; it provides a general acknowledgment.
 - **2xx—Success.** The request was successfully **handled** and **processed**.
 - **3xx—Redirection.** The browser must follow the provided link to allow the user to log in, for example.
 - **4xx—Client error.** A problem occurred with the request. The request sent invalid data, for example, or the user isn't authorized to perform the request.
 - **5xx—Server error.** A problem on the server caused the request to fail.

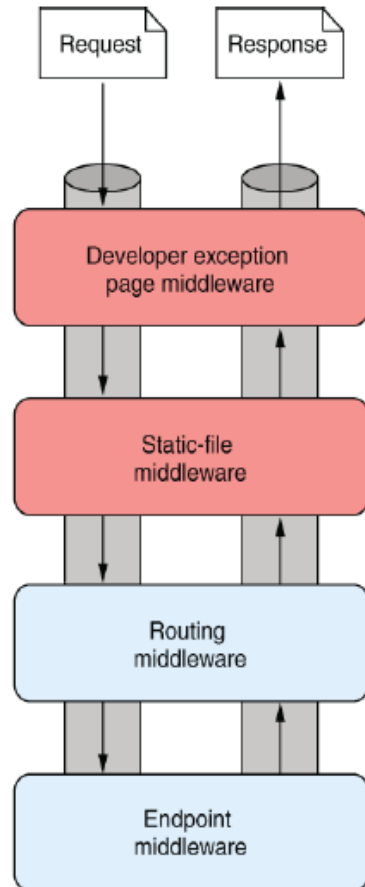
COMBINING MIDDLEWARE IN A PIPELINE — A MINIMAL API APPLICATION

The developer exception page middleware was added first, so it is the first (and last) middleware to process the request.

The static-file middleware is the second middleware in the pipeline. It handles requests for static files before they get to the endpoint middleware.

The routing middleware attempts to find an endpoint that will handle the request.

The endpoint middleware is the last in the pipeline. If there is no endpoint to handle the request, the pipeline returns a 404 response.



```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);  
WebApplication app = builder.Build();
```

```
UseDeveloperExceptionPage();           ❶  
app.UseStaticFiles();                   ❷  
app.UseRouting();                       ❸  
  
app.MapGet("/", () => "Hello World!");  ❹  
  
app.Run();
```

- ❶ This call isn't strictly necessary, as it's already added by WebApplication by default.
- ❷ Adds the `StaticFileMiddleware` to the pipeline
- ❸ Adds the `RoutingMiddleware` to the pipeline
- ❹ Defines an endpoint for the application

COMBINING MIDDLEWARE IN A PIPELINE — A MINIMAL API APPLICATION

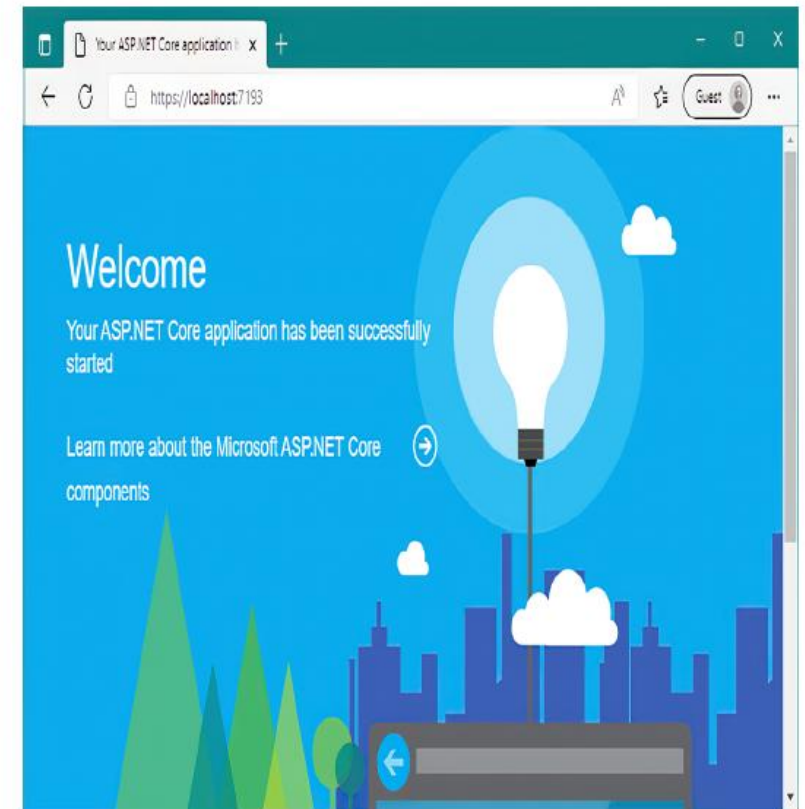
```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
WebApplication app = builder.Build();

app.UseWelcomePage("/");           ❶
app.UseDeveloperExceptionPage();
app.UseStaticFiles();
app.UseRouting();                 ❷

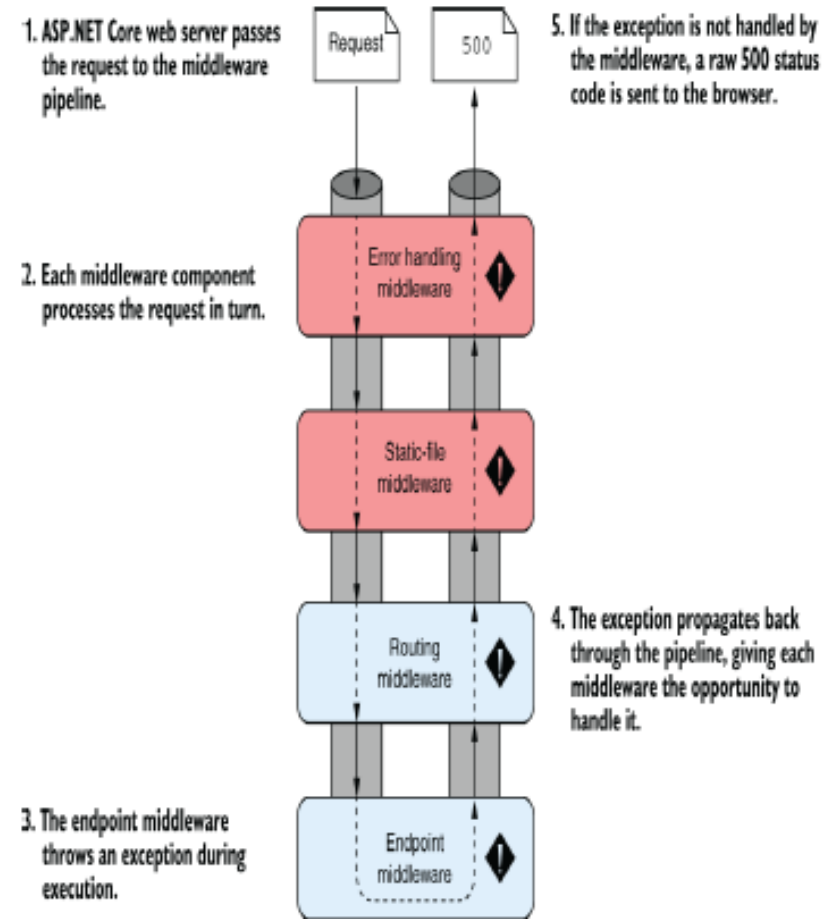
app.MapGet("/", () => "Hello World!"); ❷

app.Run();
```

- ❶ `WelcomePageMiddleware` handles all requests to the `/` path and returns a sample HTML response.
- ❷ Requests to `/` will never reach the endpoint middleware, so this endpoint won't be called.



HANDLING ERRORS USING MIDDLEWARE



VIEWING EXCEPTIONS IN DEVELOPMENT: DEVELOPEREXCEPTIONPAGE

```
app.UseDeveloperExceptionPage();
```

Title indicating the problem

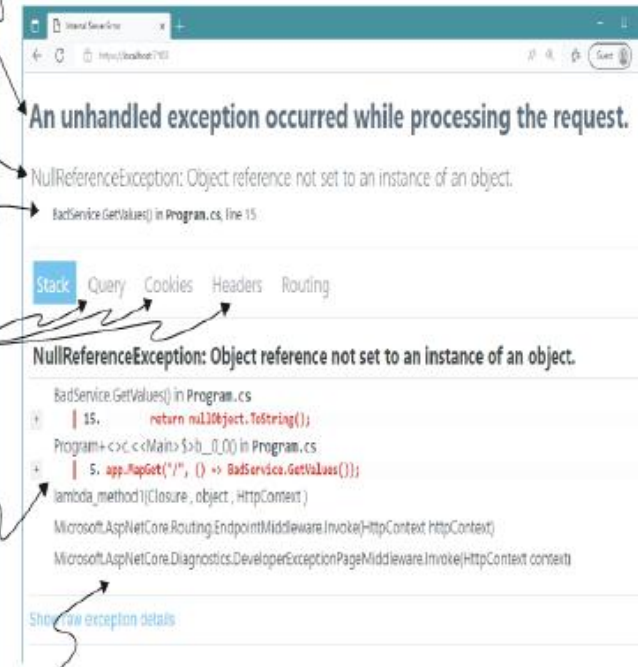
Detail of the exception that occurred

Location in the code where exception occurred

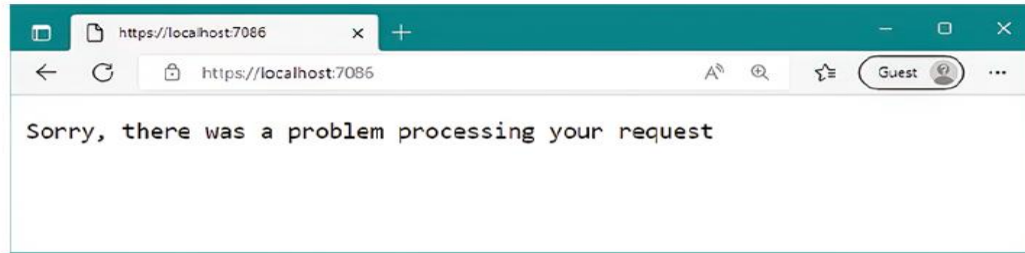
Buttons to click that reveal further details about the request that caused the exception

Code that caused the exception. You can click the "+" symbol to expand the code around the exception.

Full stack trace for the exception

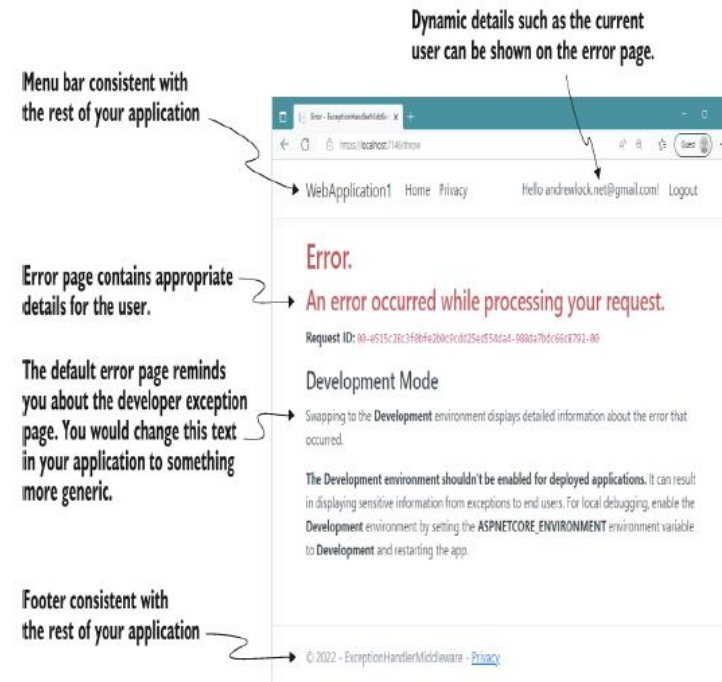


HANDLING EXCEPTIONS IN PRODUCTION: EXCEPTIONHANDLERMIDDLEWARE



```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);  
WebApplication app = builder.Build();  
  
if (!app.Environment.IsDevelopment())  
{  
    app.UseExceptionHandler("/error");  
}  
  
// additional middleware configuration  
app.MapGet("/error", () => "Sorry, an error occurred");
```

- ❶ In development, WebApplication automatically adds the developer exception page middleware.
- ❷ Configures a different pipeline when not running in development
- ❸ The ExceptionHandlerMiddleware won't leak sensitive details when running in production.
- ❹ This error endpoint will be executed when an exception is handled.



HANDLING EXCEPTIONS IN PRODUCTION: EXCEPTIONHANDLERMIDDLEWARE

