Part 10: Logging and Error Handling

Step 1: Install Logging Packages

* Action: Install Serilog for logging in your .NET Core project.
* Commands:

dotnet add package Serilog.AspNetCore

dotnet add package Serilog.Sinks.Console

dotnet add package Serilog.Sinks.File

Step 2: Configure Serilog

* Action: Update Program.cs to include Serilog configuration.
* Code:

using Serilog;

var builder = WebApplication.CreateBuilder(args);

Log.Logger = new LoggerConfiguration()

.WriteTo.Console()

.WriteTo.File("logs/log-.txt", rollingInterval: RollingInterval.Day)

.CreateLogger();

builder.Host.UseSerilog();

// ... additional configuration

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseSerilogRequestLogging(); // Log HTTP requests

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

Step 3: Create Global Error Handling Middleware

* Action: Develop a middleware class for centralized exception handling.
* File: Middlewares/ErrorHandlerMiddleware.cs

using Microsoft.AspNetCore.Http;

using Serilog;

using System.Net;

using System.Text.Json;

namespace ECommerceAPI.Middlewares

{

public class ErrorHandlerMiddleware

{

private readonly RequestDelegate \_next;

public ErrorHandlerMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task Invoke(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

await HandleExceptionAsync(context, ex);

}

}

private static Task HandleExceptionAsync(HttpContext context, Exception ex)

{

Log.Error(ex, "An error occurred while processing the request.");

var response = context.Response;

response.ContentType = "application/json";

response.StatusCode = (int)HttpStatusCode.InternalServerError;

var result = JsonSerializer.Serialize(new

{

Message = "An unexpected error occurred.",

Details = ex.Message

});

return response.WriteAsync(result);

}

}

}

* Register Middleware: Add this in Program.cs:

app.UseMiddleware<ECommerceAPI.Middlewares.ErrorHandlerMiddleware>();

Step 4: Use Logging in Controllers

* Action: Implement logging in controller actions, particularly in ProductsController.
* Example:

[HttpPost]

public async Task<ActionResult<Product>> PostProduct(Product product)

{

try

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

Log.Information("Product {ProductName} created with ID {ProductId}", product.Name, product.Id);

return CreatedAtAction(nameof(GetProduct), new { id = product.Id }, product);

}

catch (Exception ex)

{

Log.Error(ex, "Failed to create product {ProductName}", product.Name);

throw; // Let the error middleware handle the exception

}

}

Step 5: Add Detailed Error Responses

* Action: Enhance error messages in the controller.

if (string.IsNullOrEmpty(product.Name))

{

Log.Warning("Product creation failed due to missing name.");

return BadRequest(new { Message = "Product name is required.", ErrorCode = "ERR\_MISSING\_PRODUCT\_NAME" });

}

Step 6: Test the Error Handling

* Action:
  + Trigger errors by:
    - Invalid Data: POST to /api/Products with an invalid JSON:

{}

- \*\*Non-existent Category:\*\* POST with:

{

"Name": "Test Product",

"Description": "This is a test product.",

"Price": 19.99,

"Stock": 100,

"CategoryId": 999 // Assume 999 does not exist

}

* Throwing an Exception: Temporarily add throw new Exception("Test exception"); in a controller method.
* Expected Result:
  + Verify the structured error responses:
    - For invalid data: 400 Bad Request with "Product name is required."
    - For non-existent category: 400 Bad Request or equivalent with a message about category not existing.
    - For exceptions: 500 Internal Server Error with "An unexpected error occurred."
  + Check for corresponding log entries in console and log files.

Step 7: Future Enhancements

* Considerations:
  + Integrate with external logging services like Seq or Azure Application Insights for better monitoring.
  + Implement custom error codes for more precise error handling and client-side processing.