Got it! Let’s break this down step by step and outline a complete solution for your **ASP.NET Core WebAPI Online Bookstore** project with **log4net integration**. I’ll provide the structure, key code, and logging setup.

**1. Project Setup**

1. **Create the WebAPI project**:

dotnet new webapi -n dotnetapp

cd dotnetapp

1. **Configure the application to run on port 8080**:  
   Edit Properties/launchSettings.json:

"applicationUrl": "http://0.0.0.0:8080"

1. **Restore and run the project**:

dotnet restore

dotnet build

dotnet run

**2. Install EF Core Tools (Optional for Database)**

dotnet new tool-manifest

dotnet tool install --local dotnet-ef --version 6.0.6

dotnet dotnet-ef

**For initial DB setup** (if using EF):

dotnet dotnet-ef migrations add InitialSetup

dotnet dotnet-ef database update

**3. Install log4net**

dotnet add package log4net

**Configure log4net.config** in project root:

<?xml version="1.0" encoding="utf-8" ?>

<log4net>

<root>

<level value="DEBUG" />

<appender-ref ref="ConsoleAppender" />

<appender-ref ref="RollingFileAppender" />

</root>

<appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date %-5level %logger - %message%newline" />

</layout>

</appender>

<appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">

<file value="Logs/bookstore.log" />

<appendToFile value="true" />

<rollingStyle value="Size" />

<maxSizeRollBackups value="5" />

<maximumFileSize value="5MB" />

<staticLogFileName value="true" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date %-5level %logger - %message%newline" />

</layout>

</appender>

</log4net>

**Configure log4net in Program.cs**:

using log4net;

using log4net.Config;

using System.Reflection;

var builder = WebApplication.CreateBuilder(args);

// Configure log4net

var logRepository = LogManager.GetRepository(Assembly.GetEntryAssembly());

XmlConfigurator.Configure(logRepository, new FileInfo("log4net.config"));

builder.Services.AddControllers();

var app = builder.Build();

app.MapControllers();

app.Run();

**4. Book Model**

Create Models/Book.cs:

namespace dotnetapp.Models

{

public class Book

{

public int Id { get; set; } // auto-incremented

public string Title { get; set; }

public string Author { get; set; }

public decimal Price { get; set; }

}

}

**5. Book Controller with log4net**

Create Controllers/BookController.cs:

using log4net;

using Microsoft.AspNetCore.Mvc;

using dotnetapp.Models;

namespace dotnetapp.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class BookController : ControllerBase

{

private static readonly ILog log = LogManager.GetLogger(typeof(BookController));

private static List<Book> books = new List<Book>

{

new Book{ Id=1, Title="C# in Depth", Author="Jon Skeet", Price=499.99M },

new Book{ Id=2, Title="ASP.NET Core Guide", Author="Microsoft", Price=299.99M },

new Book{ Id=3, Title="Entity Framework Core", Author="Julie Lerman", Price=399.99M }

};

[HttpGet]

public IActionResult Index()

{

log.Info("Homepage requested. Returning all books.");

return Ok(books);

}

[HttpGet("search")]

public IActionResult Search(string title)

{

try

{

var results = books.Where(b => b.Title.Contains(title, StringComparison.OrdinalIgnoreCase)).ToList();

if (results.Any())

{

log.Info($"Search successful for: {title}");

return Ok(results);

}

else

{

log.Warn($"No books found for search: {title}");

return NotFound("No books found");

}

}

catch (Exception ex)

{

log.Error("Error during search", ex);

return StatusCode(500, "Internal server error");

}

}

[HttpGet("{id}")]

public IActionResult Details(int id)

{

var book = books.FirstOrDefault(b => b.Id == id);

if (book != null)

{

log.Info($"Details requested for book ID: {id}");

return Ok(book);

}

log.Warn($"Book with ID {id} not found");

return NotFound();

}

[HttpPost("purchase/{id}")]

public IActionResult Purchase(int id)

{

var book = books.FirstOrDefault(b => b.Id == id);

if (book != null)

{

log.Info($"Book purchased: {book.Title}, ID: {id}");

return Ok($"Purchase successful for {book.Title}");

}

log.Warn($"Purchase attempt failed. Book ID {id} not found");

return NotFound();

}

}

}

**6. Test the Application**

**Endpoints:**

1. GET /api/book → List all books
2. GET /api/book/search?title=Core → Search books by title
3. GET /api/book/{id} → Get details of a book
4. POST /api/book/purchase/{id} → Purchase a book

**Check logs**:  
Logs/bookstore.log will contain all info, warn, and error messages generated by user actions.

This setup covers:

* A working **BookController**
* A **Book model**
* **log4net integration** for info, warnings, and errors
* Simple in-memory book data (can be replaced with EF Core & SQL Server)

I can also provide a **version with EF Core integration and database logging** so searches, purchases, and books are persisted in SQL Server.

Do you want me to include that?