**.Net Core Clean Architecture.**

If you're working with .NET Core projects, it's essential to ensure compatibility with the frameworks, libraries, and tools you use. Here's a general set of steps for managing your .NET Core environment:

**1. Verify Installed .NET SDKs:**

Check all installed .NET Core SDK versions on your machine:

dotnet --list-sdks

This command lists all installed .NET Core SDK versions.

**2. Managing SDK Versions:**

Install SDK: If you need a specific SDK version not installed, you can get it from the .NET download page.

dotnet-sdk-version-manager install <SDK\_VERSION>

Switch SDK Version: You can switch between installed SDK versions.

dotnet-sdk-version-manager use <SDK\_VERSION>

**3. Check .NET Core Runtime Versions:**

Check installed .NET Core runtimes:

dotnet --list-runtimes

**4. Updating .NET Core SDK:**

Automatic Updates (Windows): On Windows, the .NET SDK might update automatically. You can disable or configure automatic updates through the installer.

Manual Update (Mac/Linux/Windows): For manual updates, visit the .NET download page and install the latest SDK version.Here are some general guidelines:

**ASP.NET Core and Clean Architecture:**

If you're using ASP.NET Core and implementing Clean Architecture, ensure your ASP.NET Core version is compatible with the .NET Core SDK version you're using.

Also, check the version compatibility of any third-party libraries or packages you're using within your Clean Architecture implementation.

**Library/Framework Compatibility:**

If you're using specific libraries or frameworks within your Clean Architecture project (e.g., Entity Framework Core, MediatR, etc.), check their documentation or release notes for compatibility with the .NET Core SDK version you have installed.

**Maintaining Consistency:**

It's generally a good practice to keep your .NET Core SDK version consistent across your development, staging, and production environments to avoid unexpected issues related to compatibility.

**NLog Implementation In Clean Architecture.**  
Certainly! Setting up a custom middleware for global logging using NLog in a Clean Architecture with a database involves several steps. Below is a step-by-step guide to help you achieve this in an ASP.NET Core application.

**1. Install NLog Packages**

First, install the required NuGet packages for NLog:

dotnet add package NLog

dotnet add package NLog.Web.AspNetCore

**2. Configure NLog in appsettings.json**

Set up NLog configuration in your **appsettings.json**:

{

"Logging": {

"LogLevel": {

"Default": "Information"

},

"NLog": {

"LogLevel": {

"Default": "Info",

"Microsoft": "Warn",

"Microsoft.Hosting.Lifetime": "Information"

}

}

}

Create a table in your database to store logs.

**4. Configure NLog in NLog.config**

Create an **NLog.config** file at the root of your project:

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

<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<targets>

<target xsi:type="Database" name="databaseTarget" connectionString="${var:DbConnection}" commandText="INSERT INTO LogTable(LogLevel, Message, CreatedAt) VALUES (@logLevel, @message, @createdAt)">

<parameter name="@logLevel" layout="${level}" />

<parameter name="@message" layout="${message}" />

<parameter name="@createdAt" layout="${date}" />

</target>

</targets>

<rules>

<logger name="\*" minlevel="Info" writeTo="databaseTarget" />

</rules>

</nlog>

**5. Create Logging Middleware**

Create a logging middleware that uses NLog to log requests to the database.

// LoggingMiddleware.cs

using Microsoft.AspNetCore.Http;

using NLog;

using System;

using System.Threading.Tasks;

public class LoggingMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger \_logger = LogManager.GetCurrentClassLogger();

public LoggingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task Invoke(HttpContext context)

{

try

{

// Log request information

\_logger.Info($"Request: {context.Request.Method} {context.Request.Path}");

await \_next(context);

}

catch (Exception ex)

{

// Log exceptions

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

throw;

}

}

}

**6. Register Middleware and Services**

Register the middleware and NLog services in your **Startup.cs** or program.cs file.

// Startup.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using NLog.Extensions.Logging;

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

// Add your database context and other services

services.AddLogging(builder =>

{

builder.ClearProviders();

builder.SetMinimumLevel(Microsoft.Extensions.Logging.LogLevel.Trace);

builder.AddNLog(Configuration);

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// Other middleware setup

app.UseMiddleware<LoggingMiddleware>();

}

}

**7. Usage and Enhancements**

This middleware will log incoming requests to the database using NLog. You can enhance it to log responses, additional request details, or specific log levels as needed.

Ensure your **NLog.config** connection string points to your database correctly and adjust the log targets, layouts, and rules according to your preferences and requirements.