**Developer Exception Page in ASP.NET Core (.NET 6.0)**

The Developer Exception Page in ASP.NET Core provides detailed error information for developers during the development phase.

This page helps developers to understand the errors in their application by providing a detailed stack trace and other useful information.

**Enabling Developer Exception Page**

The Developer Exception Page is typically enabled only in the Development environment to avoid exposing sensitive information in production. You can enable it in the Startup.cs file (or Program.cs in .NET 6.0 minimal hosting model).

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{  
 app.UseExceptionHandler("/Home/Error");  
}

**Features of Developer Exception Page**

When an exception occurs, the Developer Exception Page provides:

1. **Stack Trace**: A detailed stack trace showing where the error occurred.
2. **Query String Parameters**: Any query string parameters present in the request.
3. **Cookies**: All cookies sent with the request.
4. **Headers**: HTTP headers included in the request.
5. **Routing Information**: Information about the route that was matched.
6. **Environment Variables**: A list of environment variables available to the application.

## NLOG

The nlog.config file is used to configure NLog, a logging framework for .NET applications.

It defines the **targets** where **log messages will be written**, the **rules** **for routing log messages** to **targets**, and various settings for formatting log messages.

Here's an explanation of each component in the nlog.config file:

1. **<nlog> Element**: This is the root element of the configuration file. It contains all other elements and attributes related to NLog configuration.
2. **<extensions> Element**: This element is used to specify NLog extensions that should be loaded. For example, the NLog.Web.AspNetCore extension is added here to enable ASP.NET Core layout renderers.
3. **<targets> Element**: This element is used to define the targets where log messages will be written. Each target is specified as a child element of <targets>.
   * **<target> Element**: This element defines a single target. It can have various attributes and child elements to configure the target's behavior.
     + **name Attribute**: Specifies a unique name for the target.
     + **xsi:type Attribute**: Specifies the type of the target. For example, ColoredConsole specifies a colored console target.
     + **layout Attribute**: Specifies the layout pattern for log messages written to this target.
4. **<rules> Element**: This element is used to define **rules** **for routing log messages** to **targets** based on the logger name and log level. Each rule is specified as a child element of <rules>.
   * **<logger> Element**: Defines a single rule for routing log messages. It can have the following attributes:
     + **name Attribute**: Specifies the logger name pattern to which this rule applies. The \* wildcard matches all loggers.
     + **minlevel Attribute**: Specifies the minimum log level for messages to be processed by this rule.
     + **writeTo Attribute**: Specifies the target(s) to which log messages should be written.
5. **Layout Renderer**: In the layout attribute of the <target> or <rules> elements, you can use layout renderers to customize the format of log messages. Layout renderers are placeholders that are replaced with actual values at runtime. For example, ${date} is a layout renderer that outputs the current date and time.

### Ignoring namespace, class or method

To ignore logs from a specific namespace, class, or method using NLog, you can use a combination of logger names and filters. Here's how you can do it:

Ignoring Logs from a Namespace: You can use the logger element in your nlog.config file to configure logging for a specific namespace and set the minlevel attribute to Off to disable logging for that namespace.

<logger name="Namespace.To.Ignore.\*" minlevel="Off" writeTo="target" />

Ignoring Logs from a Class or Method: You can use the logger element with a specific name to target a class or method and set the minlevel attribute to Off to disable logging for that class or method.

<logger name="Namespace.ClassName.MethodName" minlevel="Off" writeTo="target" />

Ignoring Logs from a Specific Level in a Method: If you only want to ignore logs from a specific level (e.g., Debug, Info, Warn, Error, Fatal) in a method, you can use a filter with a condition.

<logger name="\*" writeTo="target">

<filters defaultAction="Log">

<when condition="level >= LogLevel.Debug and logger == 'Namespace.ClassName.MethodName'" action="Ignore" />

</filters>

</logger>

### Filters in Nlog

In NLog, filters are used to control which log messages should be processed and written to targets based on certain conditions. Filters can be applied at various levels, such as on individual loggers, targets, or rules. They allow you to fine-tune the logging behavior of your application. Here's an overview of how filters work in NLog:

Logger Filters: Filters can be applied directly to individual loggers to control whether log messages from that logger should be processed. For example, you can set a filter to only log messages from a specific logger if they are above a certain log level.

<logger name="MyApp.\*" minlevel="Info" writeTo="file">

<filters>

<when condition="level >= LogLevel.Info" action="Ignore" />

</filters>

</logger>

Target Filters: Filters can also be applied to targets to control whether log messages should be sent to that target. This can be useful if you want to send certain log messages to a specific target based on their content.

<target name="file" xsi:type="File" fileName="log.txt">

<filters>

<when condition="contains('${message}', 'error')" action="Log" />

</filters>

</target>

Rule Filters: Filters can be applied to rules to control whether log messages should be routed to specific targets based on certain conditions. This allows you to customize the routing of log messages based on their content, log level, or other properties.

<rules>

<logger name="\*" minlevel="Trace" writeTo="file">

<filters>

<when condition="level >= LogLevel.Info" action="Ignore" />

</filters>

</logger>

</rules>

Filters in NLog use a condition attribute to define the condition that must be met for the filter to take action. The action attribute specifies what action should be taken if the condition is met, such as Log to log the message or Ignore to ignore it. Filters provide a flexible way to control the logging behavior of your application and can be used to implement custom logging logic based on your requirements.