Logging

ASP .NET Core

Yash Lathiya

Table of Contents

Logging API

- **Purpose**:
  Provides a framework for logging application messages and events, aiding in diagnostics and monitoring.
  It helps developers track the flow of the application and troubleshoot issues effectively.

- **ILogger Interface**:
  logging messages at various levels (e.g., Trace, Debug, Information, Warning, Error, Critical).

- **ILoggerFactory and ILoggerProvider**:
  ILoggerFactory is responsible for creating ILogger instances, while ILoggerProvider defines how and where the log messages are stored or displayed.

- **Dependency Injection**:
  built-in Dependency Injection (DI) framework.

- **Logging Levels**: To categorize logs

  Trace: For very detailed logs, typically only valuable during development.

  Debug: Information useful during development and debugging.

  Information: General operational messages about the application's normal functioning.

  Warning: Indications of possible issues or significant events that are not errors.

  Error: Errors and exceptions that should be addressed.

  Critical: Critical errors causing a major failure.

- **Configuration**: Logging behaviour can be configured through appsettings.json or other configuration files, enabling different logging settings for different environments (e.g., development, staging, production).

- **Structured Logging**: The API supports structured logging, allowing developers to log complex objects and include context information, which makes the logs more informative and easier to query and analyze.

Logging Providers

- **Overview**:
  NLog is a popular third-party logging framework for .NET applications, known for its flexibility and performance.
  It supports a wide range of logging targets, including files, databases, console, email, and more.

- **Integration with ASP.NET Core**:
  NLog can be easily integrated with ASP.NET Core using the NLog.Extensions.Logging package.
  This integration allows developers to configure NLog through XML or JSON configuration files and leverage its powerful features within ASP.NET Core applications.

- **Configuration**:
  NLog configuration involves setting up targets (where logs are sent) and rules (which logs are sent to which targets).
  The configuration is typically done in an NLog.config file, but it can also be done programmatically.

- **Features**:
  NLog offers advanced features such as asynchronous logging, buffered logging, log filtering, and layout rendering. These features provide greater control over logging behavior and performance.