SOLUTION APPROACH DOCUMENT

LOGGING FRAMEWORK USING .NET 6.0

Report Submitted by

PULIVENDULA ANUHYA (VTU17129)

EJASWANTH (VTU18304)

DHARMENDRA KUMAR (VTU17027)

AABHINAYA (VTU17637)

SYED RAFIYA ZAKKI (VTU17054)

Under the guidance of

Mrs.P.ARIVUBRAKAN (Assistant Professor)

Dr.MURALIDHAR MS (Associate Professor)

Dr.T. KUJANI (Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN Dr.SAGUNTHALA R&D INSTITUTE OF
SCIENCE AND TECHNOLOGY

ABSTRACT

Applications that have been deployed to a production needs to be monitored. One of the best ways to monitor application behavior is by emitting, saving, and indexing log data. Logs can be sent to a various destination for indexing, where they can then be searched when problems arise. In the .NET core application, logging is a built-in feature for some types of applications such asASP.NET Core and .NET Core Work Services. The Logging API doesn't work independently; it works with one or more logging suppliers that store log messages to a destination place. This application is created for file log and database log. These log activities such as Debug, Info and Error are stored. This framework is developed using ASP.Net which is reusable and easy to monitor the dot net applications.

Keywords: ASP .NET, API, Log.

LIST OF FIGURES

ABBREVIATIONS

ASP	Active Server Pages
API	Application User Interface
XML	Extensible Markup Language
JSON	Java Script Object Notation
SQL	Structured Query Language
HTTP	Hyper Text Transfer Protocol

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

1.1 ABOUT THIS DOCUMENT

Logging is essential for a software development, but it's often disregarded until the program crashes. Logging serves numerous purposes, including root cause and bug analysis, as well as performance reviews for the application. The .NET developer can write logs to several locations besides a flat file. The C# and VB.NET languages offer internal libraries to help While they're limited, you can find several free, easy-to-use frameworks with some extended capabilities, which we'll discuss a few of them here.NET has a pluggable logging architecture using ILoggerProvider to control where the log messages are written.

1.1.1 PURPOSE & SCOPE

The purpose of our application is to provide a structured and efficient way to capture and record log information during application execution, aiding in troubleshooting, debugging, and monitoring. The logging framework covers various aspects, including centralized logging, customizable logging levels, support for multiple logging targets, structured logging, performance considerations, and its applicability in debugging, production monitoring, auditing, compliance

1.1.2 OVERVIEW

Logging is an essential part of any application, as it helps developers identify and fix errors and track system behavior..NET offers several logging frameworks that developers can use to implement logging in their applications.

One of the most popular logging frameworks in .NET is log4net. Log4net is an open-source logging library that provides a flexible and configurable logging framework for .NET applications. It supports logging to various destinations such as a console, file, database, email, and others.Log4net offers different log levels such as DEBUG, INFO, WARN, ERROR, and FATAL to categorize the logs based on their severity.

Here are some of the features of the logging framework in .NET Core 6.0:

Logging providers: The logging framework provides several built-in logging providers that allow developers to log messages to various destinations such as a console, file, event log, and others.

Log filtering: The logging framework allows developers to filter the log messages based on their log level, category, or message text.

Structured logging: The logging framework supports structured logging, which allows developers to log events unstructured format that can be easily searched, analyzed, and visualized.

Log enrichment: The logging framework allows developers to enrich the log messages with additional context such as user ID, request ID, and others.

CHAPTER 2

COMPONENT DESIGN

2.1 COMPONENT DESIGN DIAGRAM

Logging is a powerful tool for understanding and debugging program's run time behavior. Logging is the process of writing log message during the execution of a program to a central place. This logging allows you to report and persist error and warning messages as well as info messages on that the messages can later be retrieved and analyzed. The component design diagrams like work flow and sequence diagrams are given below:

2.1.1 OVERALL WORKFLOW

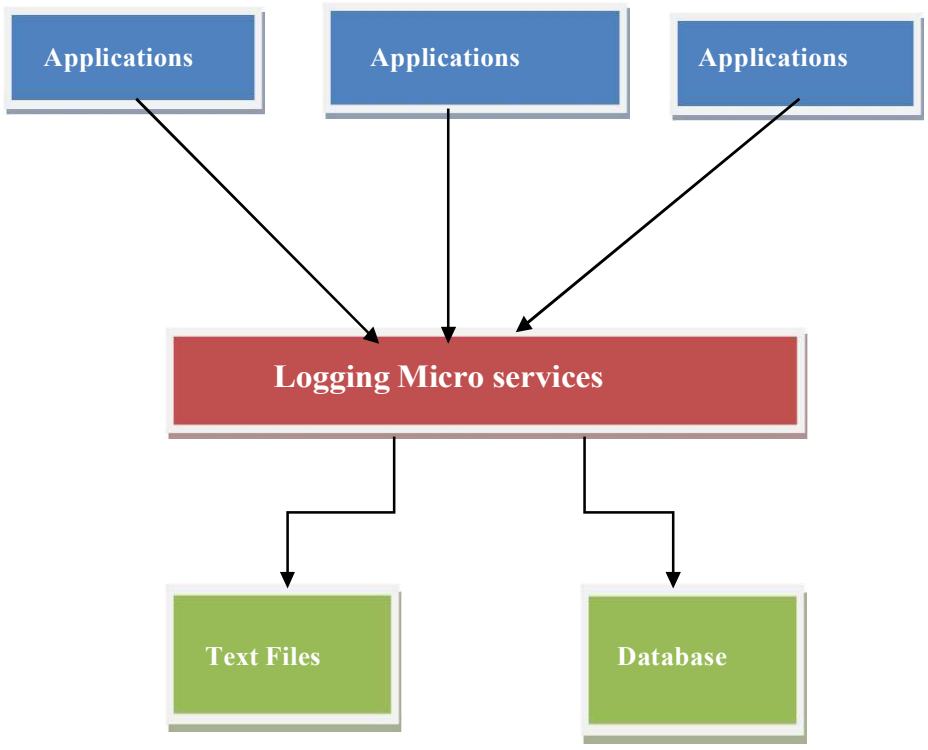

Fig2.1: *Component Design of logging framework*

- **SEQUENCE DIAGRAM**

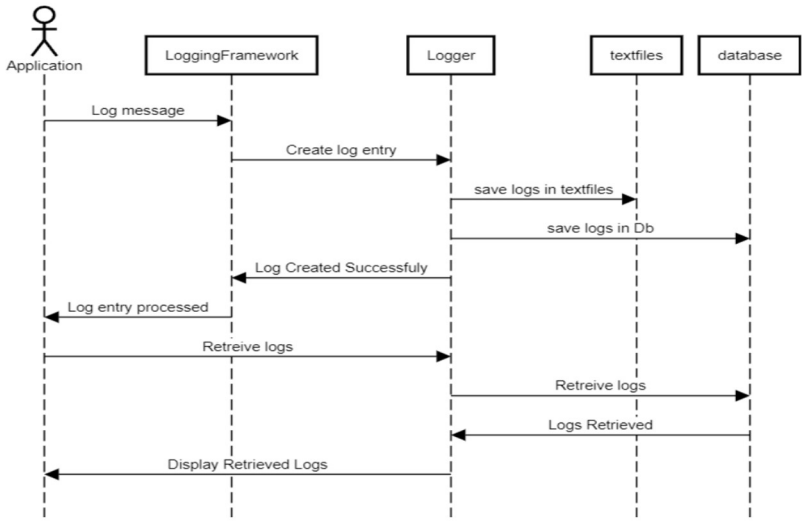

Fig2.2: *Sequence Diagram of logging framework*

1. Sequence diagram illustrates the process of logging framework with .net. The components in this process are Logging Framework ,Logger and Log retrieval.

2. Sequence diagram begins with the “Application” actor sending a “Log message” request to the “Logging Framework” component.

3. “Logging Framework” component then interacts with the “Logger” component by sending a “Create log entry” message.

4. “Logger” component receives the “Create log entry” message and performs various tasks like formatting the log message, applying log levels, filters, and other operations.

5. Once the log entry is prepared, the “Logger” component appends it to the log file.

6. After completing the log entry creation process, the “Logger” component sends a “Log entry created” message back to the “Logging Framework” component.

7. Finally, the “Logging Framework” component informs the “Application” actor that the log entry has been processed by sending a “Log entry processed” message.

2.1.2 LOW LEVEL DESIGN

Almost in all applications logging functionality is needed, but requirements vary fromproject to project. So, a Configurable Logging framework is designed.

Functional Requirements:

It should provide,

- A configurable option for other application modules to save logs at more than one platform like, on Console, in text files or on network etc.
- A Facility to Log messages in different categories like, ERROR, WARNING, GENERAL MESSAGES and also provision to control each category independently.
- A Facility to configure & bind category and Logging platform at runtime i.e. user will be able to specify at runtime that,
- Messages of any particular category should be logged or not etc.
- Messages of any particular category like ERROR should be logged in error. Txt and remaining categories on console only etc.

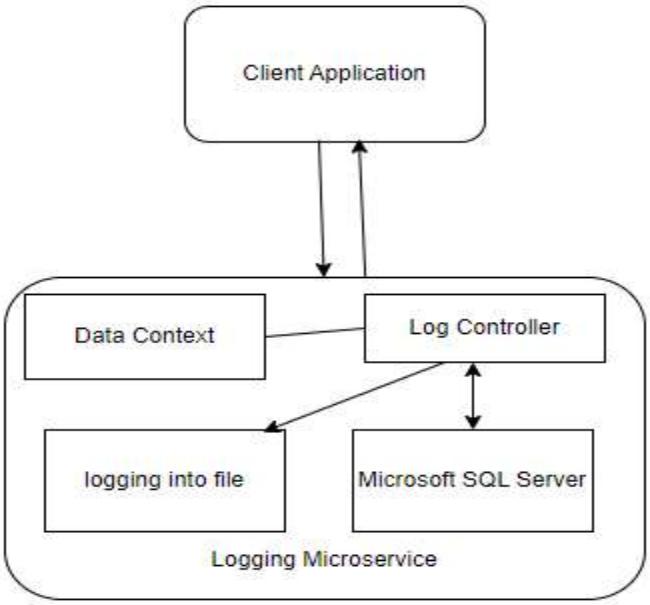

Fig2.3: *Low Level Design*

2.1.3 CLASS DIAGRAM

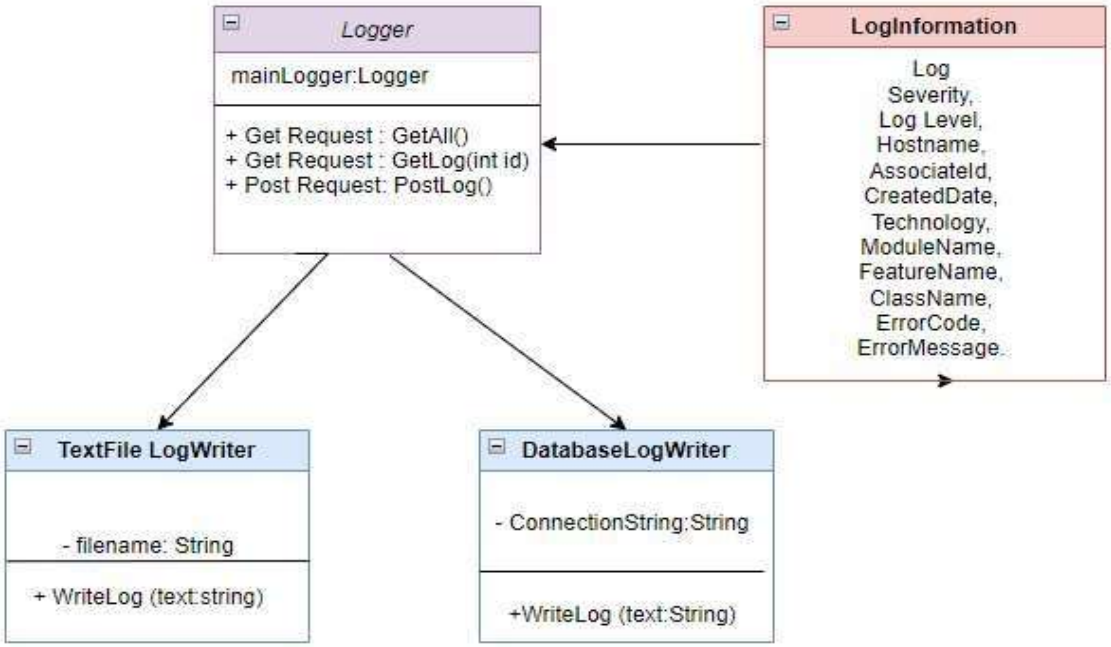

Fig2.4: *Class Diagram of Logging framework*

The components of class diagram work together to capture and store log entries. Log Information holds the log details, Logger handles logging operations and methods, TextFile Log Writer writes logs to a text file, and DatabaseLogWriter writes logs to a database using the connection string.

Log Information:

- Represents a structure or object that holds information about a log entry, including the log message, severity, log level, hostname, associate ID, created date, technology, module name, feature name, class name, error code, and error message.

Logger:

- Acts as a central component responsible for logging operations.
- Provides methods such as Get All() to retrieve all logs, GetLog(int id) to retrieve a specific log, and PostLog() to create a new log entry.

Text File Log Writer:

- Writes log messages to a text file specified by the filename.
- Provides a Write Log() method to append log messages to the text file.

Database LogWriter:

- Writes log messages to a database using the provided connection string.
- Offers a WriteLog() method to insert log messages into the appropriate database table or collection.

CHAPTER 3

TECHNOLOGY & FRAMEWORK

Tools and Languages used to develop the Logging framework for .NET core applications

3.1 .NET 6.0

- .NET is an open-source platform for building desktop, web and mobile applications that can run natively on any operating system.
- The .NET system includes tools, libraries , and languages that support modern , scalable , and high-performance software development.

3.2 JSON

- It is a text-based way of representing JavaScript object literals, arrays, and scalar data. JSON is relatively easy to read and write, while also easy for software to parse and generate.
- It is often used for serializing structured data and exchanging it over a network , typically between a server and web applications.

3.3 C#

- C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in .NET.
- C# has its roots in the C family of languages and will be immediately familiar to C, C++ , Java, and JavaScript programmers.

3.4 VS code for editing

•Visual studio code is a code editor with all languages support functionality. It supports development processes like debugging, version control, and testing.

•No separate framework or packages are needed, it can automatically plug in all libraries and frameworks.

CHAPTER 4

SOLUTION APPROACH

4.1 APPROACH DESCRIPTION

Applications that have been deployed to production must be monitored. Now a days every application contains logging libraries for recording the activities happening in the applications. One of the best ways to monitor application behavior is by emitting, saving, and indexing log data. Logs can be sent to a variety of applications for indexing, where they can then be searched when problems arise.

Separating the concerns of log management from logging simplifies application code. However, as developer must still write logging code. Effective logging can make an application highly supportable. Poor logging can make it a night mare for operations teams. It's important for developers to know what data to log, and to use patterns for logging that can been forced across an application.

In our LOGGING FRAMEWORK typically support feature including:

- **Logging levels**
- **Logging targets**
- **Structured Logging**

There are many Logging Libraries for .Net applications with different logging levels like Nlog,Log4NET apart from this we are going to develop our own Logging library called Logging micro services for .NET application.

In our Logging framework it supports the following Logging level:

- **DEBUG**: Additional information about application behavior for cases when that information is necessary to diagnose problems

- **INFO**: Application events for general purposes

- **WARN**: Application events that may be an indication of a problem

- **ERROR**: Typically logged in the catch block a try/catch block, includes the exception and contextual data

These are the logging levels which we are going to include in our build-in logging framework. Logging levels are used to filter log data. A typically logs production environment may be configured to log only error levels but in some times the logging level can be increased to include DEBUG, INFO and WARN. In our logging framework we are going to include these logging levels also.

In Future, using this framework we will integrate with multiple applications. Maintaining all application logs in a single database and file we will have application wise split. No Need to rewrite duplicate code in each project for capturing logs. Maintain logs at one place. Easy to Maintain, and splitting all logs with application wise.

4.2 Steps

- Define the requirements for our logging framework. We consider aspects such as the types of log messages to support (e.g., information, warning, error), log levels, log destinations (e.g., file, database, console), and any specific functionalities or features needed.
- Determined the structure of a log entry. Typically, it includes information such as timestamp, log level, message, source, and any additional contextual information. We Designed an appropriate class or structure to represent the log entry.
- Provide configuration options to allow users to customize the logging behavior. These options may include log level thresholds, log destinations, log file path, log format, and any other relevant settings. We Consider a configuration file or a configuration object to store these options.
- Create an interface that defines the operations for logging, such as writing log entries, setting log levels, and configuring the logger. This interface will serve as the contract for the logger implementation.

`

- Create a concrete implementation of the logger interface for various log destinations, such as a file logger, a database logger, or a console logger.
- Develop a logging framework that acts as a central component for managing log messages. The framework should provide an API for application developers to log messages easily. It will handle the configuration, routing of log entries to appropriate loggers, and any additional functionalities like filtering, formatting, or asynchronous logging.
- Integrate the logging framework into our applications. This typically involves adding the necessary references, initializing the logger, and using the provided API to log messages at relevant points in the application's code.
- Implement error handling and exception logging within the logging framework itself. Capture and log any errors that occur during the logging process to ensure robustness and prevent infinite loops or other issues.

4.3 Test Cases

1. The module is reusable and easily integrated into multiple applications.

- The module successfully connects to the database when integrated into various application environments.

- The module provides appropriate error notifications when faced with an incorrect or non-existent database connection.

- Within a fully integrated application environment, the module operates as expected, demonstrating correct logging functionality.

- The module has been designed for reusability. It integrates seamlessly into other applications without requiring any modifications.

2. Logs are captured in a standard, non-application-specific format, covering only common fields for all applications. Any additional required information is captured in separate fields.

- The module reliably adds entries to the database in a standard format.

- When all necessary log fields are provided in a request, the module correctly processes and logs the information.

- If any log fields are missing, the module identifies this situation and provides an appropriate error response.

- The module features robust exception handling. If an exception occurs during logging, the module catches it and returns an appropriate error response.

- If the database becomes unreachable during logging, the module handles this scenario gracefully.

- Upon receiving a request to retrieve all logs, the module returns the correct data, encompassing all logs present in the database.

- If no logs are present in the database, the module manages this scenario appropriately when a request is made to retrieve all logs.

- When a request is made to retrieve a single log by ID, the module returns the correct data.

- If a log with a specific ID does not exist in the database, the module handles this scenario appropriately.

- The module ensures that the logs contain all necessary standard fields.

- If there are additional fields to be logged, the module correctly adds these to the respective fields.

- The module writes the log entry to a file correctly.

- The module manages situations where the directory or file path doesn't exist, either by creating the required directories and files or providing an appropriate response.

- The module is designed to create a new file for each day's logs, ensuring an organized logging system.