**Job Logger Code Review FeedBack**

**Introduction**

In this document, I detail all the errors found on the given code, plus the suggestions for implementing best practices.

In three sections, there are explained the compile-time errors, runtime errors, and a few logical errors. At the end it can be found a section detailing the suggestions mentioned above.

It is important to note that only by fixing the errors, the code would actually work. Given the context in wich this exercise is presented, I decided to implement design patterns to the final solution.

The JobLogger class includes six static attributes which are set by the class constructor, this will give us problems because the static attributes will be shared by all the Instances in the application. We should transform the static attributes to Instances attributes removing the static keyword. Furthermore, the LogToDatabase attribute should be renamed to \_logToDatabase and the \_initialized attribute should be removed from the class because it is not used or referenced on the class.

All the six boolean attributes should be replaced by two Enum types one to Store the Log Type (Console, File and Database) and another for the Message Severity (Message, Warning and Error.)

**Method LogMessage:**

The first problem that have this method is that the first two parameters have the same name but they have different data types, string and bool.

I will assume that the second parameter has the name "isMessage".

The code message.Trim() is not validating if the string message is null, in case the message be null it will throw a NullReferenceException.

We should manage custom Exceptions by creating for example a class LoggerException which is child of the Exception Class to detect or identify that the Exception comes from the JobLogger Class.

We should avoid using variable names like "L" or "T" because those names doesn't express the meaning or purpose of the variables.

**DataBase Logging:**

We should include "usings" to avoid having long names like System.Data.SqlClient.SqlConnection and use the keyword var to reduce the code length. For example, we could use using System.Data.SqlClient;.

For the connection String we shoud use the ConfigurationManager.ConnectionStrings instead of the ConfigurationManager.AppSettings and the connection declaration should be included inside of an using keywork to close the connection when we finish to use the connection. The command is not associated with the connection, that will throw an Exception. We should use parameters on the query string to avoid SQL Injections and to keep maintainable our code.

**File Logging:**

The First condition will execute the method ReadAllText when the specified file doesn't exist, this will throw an IO Exception, furthermore, the logic is storing the File content on a string variable called l, this could cause a high memory usage or out of memory exceptions. We should use the method File.AppendText to avoid this kind of risks. For the Log file name we should use a date format like YYYYMMDD to avoid including back slashes on the Log File Name.

We should use a path variable to store the file path and reuse that variable inside the File Logger.

**Console Logging:**

We should reset the default console color after calling the Console WriteLine command to avoid keeping the last used color on the console.

**Code Refactoring**

For the code Refactoring I advice to use the Strategy Pattern because we have Loggers with different behaviors for the action LogMessage or Log (Method), to do it we have to create an Interface and specific classes to implement the Interface method LogMessage. If we need to add another Logging method we can easily create another class that implements the ILogger Interface.

In this Case the Logger class is the context of the Strategy Pattern where the SetLogType will select the Type of Logging class to use.

**Suggestions to implement good practices:**

When arranging SqlCommand CommandTexts, parameters should be passed using the SqlCommand property “parameters”.

When a log is recorded into a textfile, the entire file is read and then saved including the new line. There are framework functions to improve this by only recording a new line regardless the previous content of the text.

When logs are registered in database, there is no way to difference one from another. A column should be added, to record at least time and date of generation.

Since the different types of message are exclusive between them (a message can’t be an error and a warning at the same time), an enum would come in handy to avoid typing every time the name of the type, and for easy handling of the parameter.

**Design principles to apply:**

Given the structure of the requested method, there are various design patterns that could be easily implemented to improve the code response to changes and it’s versatility.

Protected Variations would be my first choice. There are multiple ways of recording a log described in the original statement, and each needs special treatment to successfully work. Assuming it is a strong point of variation, that can’t be parameterized, it should be treated with special attention to achieve a change - responsive solution.

Using Asynchronous Methods

In the same example, an alternative is added to record Log messages using asynchronous methods. This option accelerates the Log process.