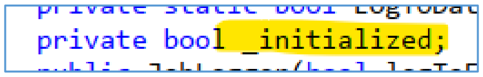
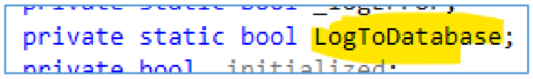
Code Review - Results

1. Delete the variable _initialized. It is not used into the project.



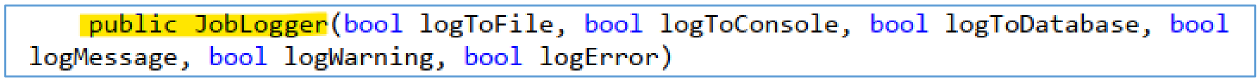
```
private static bool LogToDat
private bool _initialized;
public JobLogger(bool logToF
```

2. Since the variable LogToDatabase is a private one, use the camel case terminology.
   - Change LogToDatabase to _logToDatabase



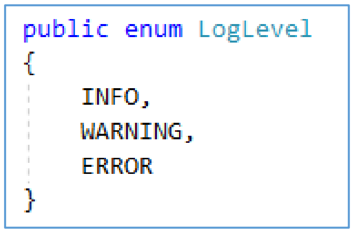
```
private static bool _logError;
private static bool LogToDatabase;
private bool _initialized;
```

3. The constructor JobLogger has too many parameters.

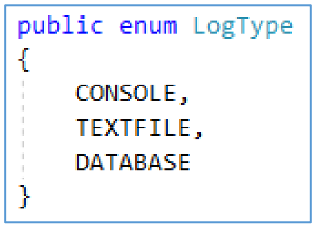


```
    public JobLogger(bool logToFile, bool logToConsole, bool logToDatabase, bool
logMessage, bool logWarning, bool logError)
```

   I recommend to define Enum types for the log type (file, console and/or database) and the log level (message/info, warning, error).

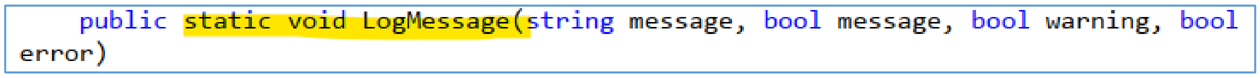


```
public enum LogLevel
{
    INFO,
    WARNING,
    ERROR
}
```



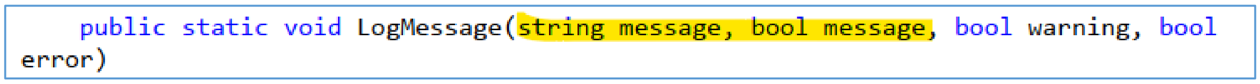
```
public enum LogType
{
    CONSOLE,
    TEXTFILE,
    DATABASE
}
```

4. Incorrect definition of static method LogMessage. You can call the method directly without previously instance creation (all the local variables _logXXX have as default value false) and the exception "Invalid Configuration" will be thrown
   - Remove the static clause from the method LogMessage



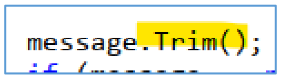
```
    public static void LogMessage(string message, bool message, bool warning, bool
error)
```

5. Duplicated parameter "message" in method LogMessage. It has two different types: string and bool.
   - Change the second one to be named info.



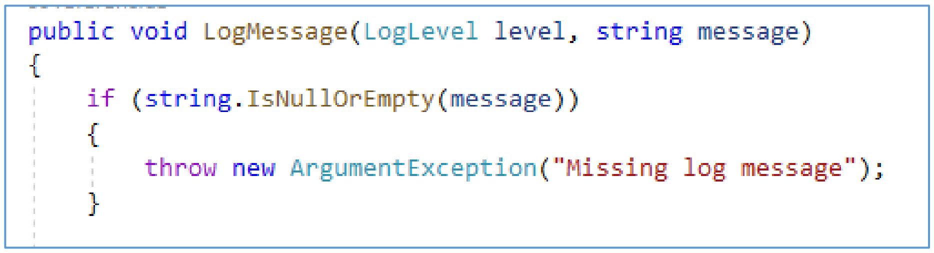
```
    public static void LogMessage(string message, bool message, bool warning, bool
error)
```

6. The code *message.Trim()* could throw an exception if the value is null.



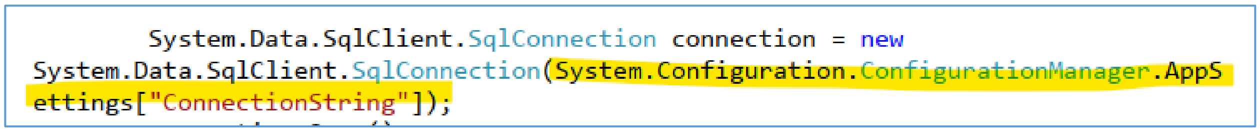
```
message.Trim();
```

   Since the message is common for all the log levels, include proper validations inside a base/abstract class. Example:

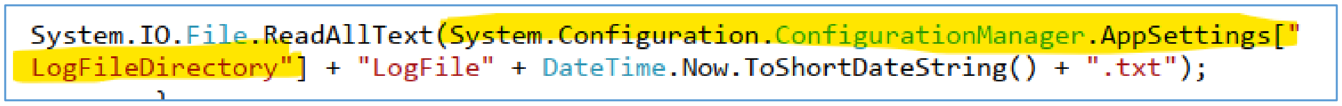


```
public void LogMessage(LogLevel level, string message)
{
    if (string.IsNullOrEmpty(message))
    {
        throw new ArgumentException("Missing log message");
    }
```

7. For security and maintenance reasons, you could use environment variables instead of config file. Change the code System.Configuration.ConfigurationManager.AppSettings["XXX"] to read from an environment variable Environment.GetEnvironmentVariable("XXX", EnvironmentVariableTarget.User)

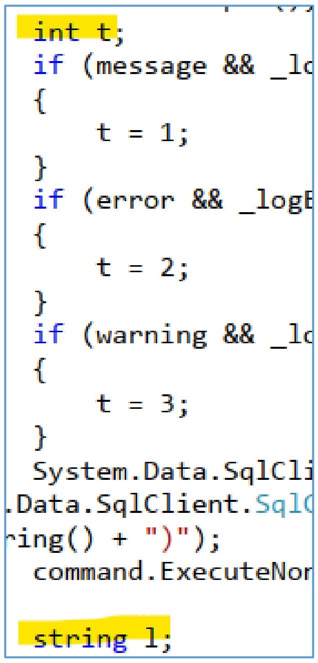


```
        System.Data.SqlClient.SqlConnection connection = new
System.Data.SqlClient.SqlConnection(System.Configuration.ConfigurationManager.AppS
ettings["ConnectionString"]);
```



```
System.IO.File.ReadAllText(System.Configuration.ConfigurationManager.AppSettings["
LogFileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt");
```

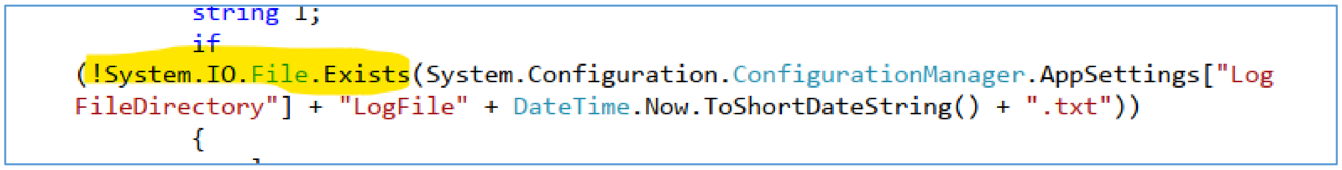
8. For debugging and maintenance purposes, use variable names with a useful meaning.
   - Change "t" to be type and "l" to be logContent.



```
int t;
if (message && _lo
{
    t = 1;
}
if (error && _logE
{
    t = 2;
}
if (warning && _lo
{
    t = 3;
}
System.Data.SqlCli
.Data.SqlClient.SqlC
ring() + ")");
command.ExecuteNon

string l;
```

9. The method ReadAllText works only if the file exits previously.
   - Change the rule *!System.IO.File.Exists* to be *System.IO.File.Exists*



```
            string l;
            if
    (!System.IO.File.Exists(System.Configuration.ConfigurationManager.AppSettings["Log
    FileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt"))
            {
```

10. The DB handling must be defined within a try-catch-finally block make sure the connection will be always closed. In addition, needs to validate is the connection string is not empty and change the *SqlCommand* to use a Stored Procedure.

```
            System.Data.SqlClient.SqlConnection connection = new
    System.Data.SqlClient.SqlConnection(System.Configuration.ConfigurationManager.AppS
    ettings["ConnectionString"]);
            connection.Open();
            int t;
            if (message && _logMessage)
            {
                t = 1;
            }
            if (error && _logError)
            {
                t = 2;
            }
            if (warning && _logWarning)
            {
                t = 3;
            }
            System.Data.SqlClient.SqlCommand command = new
    System.Data.SqlClient.SqlCommand("Insert into Log Values('" + message + "', " +
    t.ToString() + ")");
            command.ExecuteNonQuery();
```

Notice that it is also possible to use any ORM technique (like EntityFramework.NET or Dapper) to handle the database processing.

11. Incorrect use of date format for read/write the log file.

DateTime.Now.ToShortDateString() = 09/12/2019

When concatenate with the log file name will generate an error

“C:\Apps\Logger\LogFile09/12/2019.txt”

```
  System.IO.File.ReadAllText(System.Configuration.ConfigurationManager.AppSettings["
  LogFileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt");
          }
```

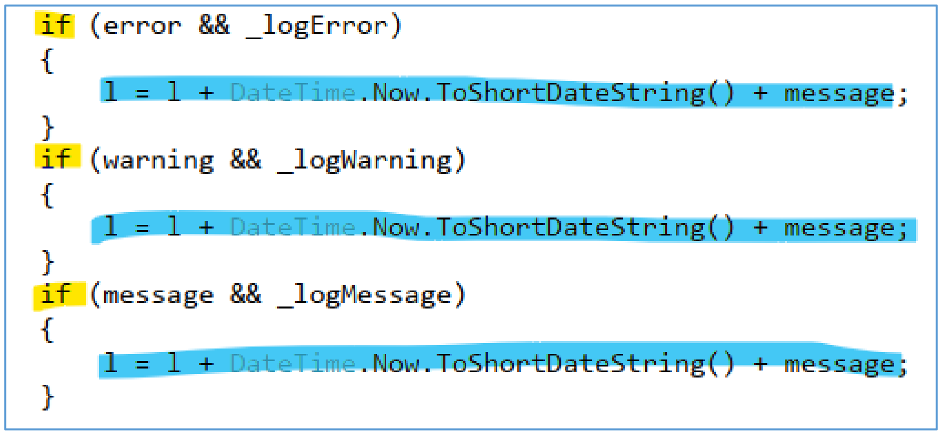
Change to use DateTime.Now.ToString(“MM-dd-yyyy”). Assign this to a local variable and use it in the read and write processes.

Validate if the *LogFileDirectory* exists before save the log. If not, create it.

- string logFileFolder = Environment.GetEnvironmentVariable("LogFileDirectory", EnvironmentVariableTarget.User)

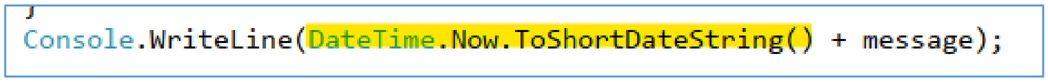
- string filePath = string.Format(@"{0}\LogFile_{1}_{2}.txt",
  logFileFolder,
  DateTime.UtcNow.ToString("MM-dd-yyy"),
  logLevelName);

12. When add a log entry in the log file use the format DateTime.Now.ToString(“HH:mm:ss”) instead of DateTime.Now.ToShortDateString() and add a carriage return at the end. Using the current format, you won’t know at what time of the day was generated the entry.
    - Use string.Format("{0}{1}: {2}\n", l, DateTime.Now.ToString("HH:mm:ss"), message)
    - In the current code: Unnecessary use of if structures. All the paths have the same result.



```
if (error && _logError)
{
    l = l + DateTime.Now.ToShortDateString() + message;
}
if (warning && _logWarning)
{
    l = l + DateTime.Now.ToShortDateString() + message;
}
if (message && _logMessage)
{
    l = l + DateTime.Now.ToShortDateString() + message;
}
```

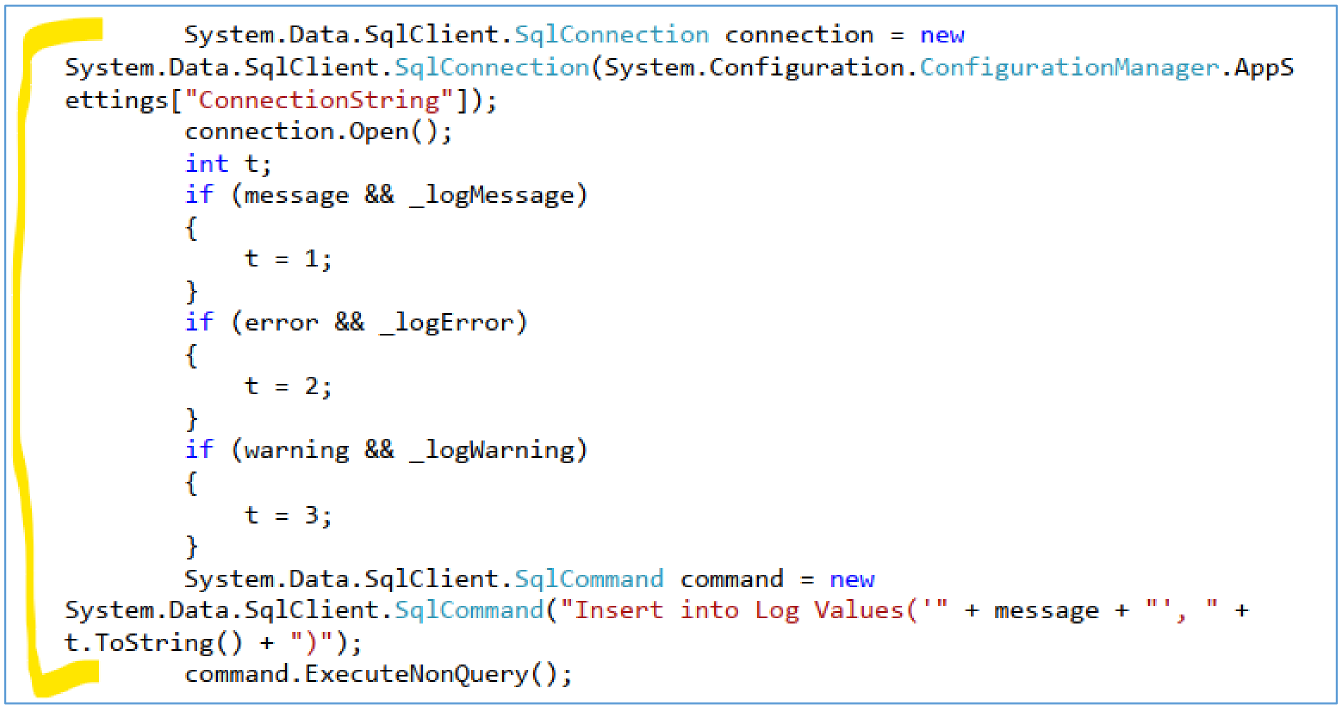
13. When add a log entry in the console use the format DateTime.Now.ToString(“MM-dd-yyyy HH:mm:ss”) instead of DateTime.Now.ToShortDateString(). Using the current format, you won’t know at what date and time of the day was generated the entry.



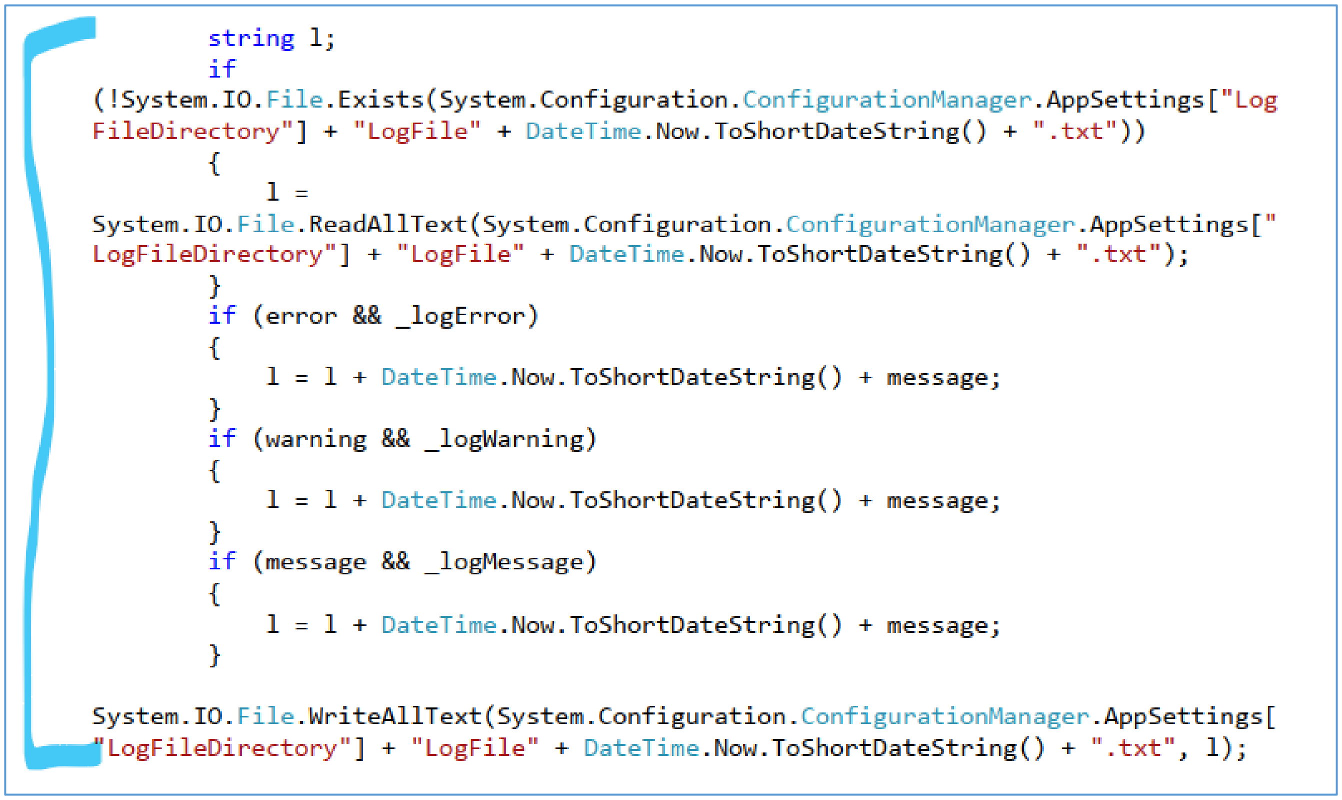
```
Console.WriteLine(DateTime.Now.ToShortDateString() + message);
```

14. The process is not checking what type of logging (database, file, console) should be applied. The variables _logFile, _logToConsole, LogToDatabase are initialized but not use properly.
    - In the current code, add conditional logic properly based on the log type, e.g.
      if (_logToDatabase) { /*yellow block here */ }
      if (_logToFile) { /*ice blue block here */ }

    - I recommend to have distinct loggers classes to manage them properly.



```
        System.Data.SqlClient.SqlConnection connection = new
System.Data.SqlClient.SqlConnection(System.Configuration.ConfigurationManager.AppS
ettings["ConnectionString"]);
        connection.Open();
        int t;
        if (message && _logMessage)
        {
            t = 1;
        }
        if (error && _logError)
        {
            t = 2;
        }
        if (warning && _logWarning)
        {
            t = 3;
        }
        System.Data.SqlClient.SqlCommand command = new
System.Data.SqlClient.SqlCommand("Insert into Log Values('" + message + "', " +
t.ToString() + ")");
        command.ExecuteNonQuery();
```



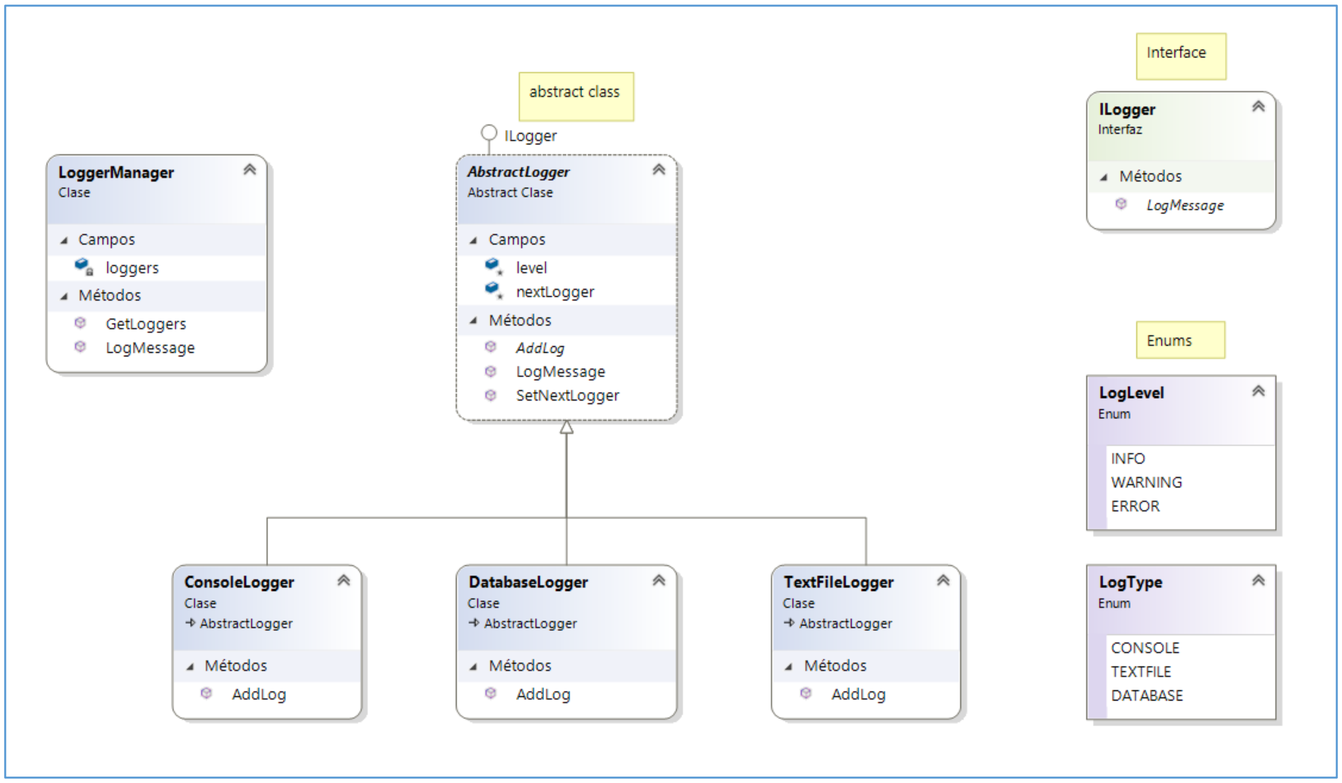
```
        string l;
        if
(!System.IO.File.Exists(System.Configuration.ConfigurationManager.AppSettings["Log
FileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt"))
        {
            l =
System.IO.File.ReadAllText(System.Configuration.ConfigurationManager.AppSettings["
LogFileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt");
        }
        if (error && _logError)
        {
            l = l + DateTime.Now.ToShortDateString() + message;
        }
        if (warning && _logWarning)
        {
            l = l + DateTime.Now.ToShortDateString() + message;
        }
        if (message && _logMessage)
        {
            l = l + DateTime.Now.ToShortDateString() + message;
        }

System.IO.File.WriteAllText(System.Configuration.ConfigurationManager.AppSettings[
"LogFileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt", l);
```

15. I recommend to use distinct log files (for messages/info, warning and error) in order to quickly review the errors and warnings for debugging and/or fixing purposes

16. I recommend to use the pattern *Chain of Responsibility* to implement different handlers to manage the logging to file, console and database.

Class Diagram for my implementation



Link for the source code: https://github.com/marvasten/Logger