1. Code review about the code provided.

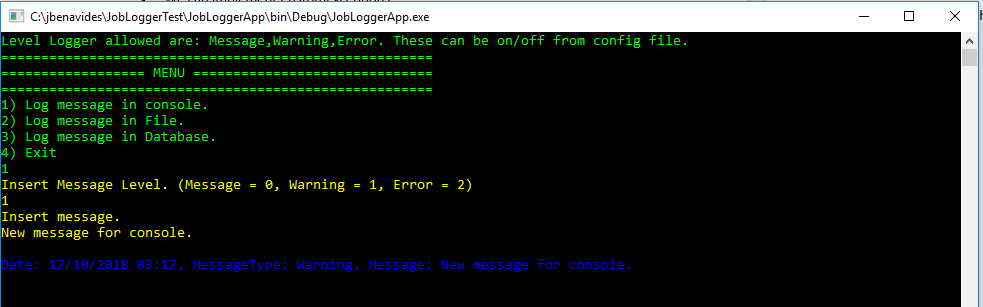
* There is no separation of concern, we should split functionality into methods and/or classes in order to be more readable and also to have a better maintainability.
* We can use string.IsnullOrEmpty() instead of check it separately.
* We can implement CustomExceptions.
* There are some variables (t) that don’t describe the value it stores, we should use variable with more descriptive names.
* Some variables names (LogToDatabase) don’t honor the code standard.
* The SqlCommand used to log to database doesn’t have a SqlConnection assigned.
* As the SqlCommand is not using a Store Procedure or SqlParameters, can suffer sql Injection.
* We should include in the DB the message type (Message, Warning, Error) plus the date.
* Should use an Enum to store the Message types.

1. JobLogger Implementation

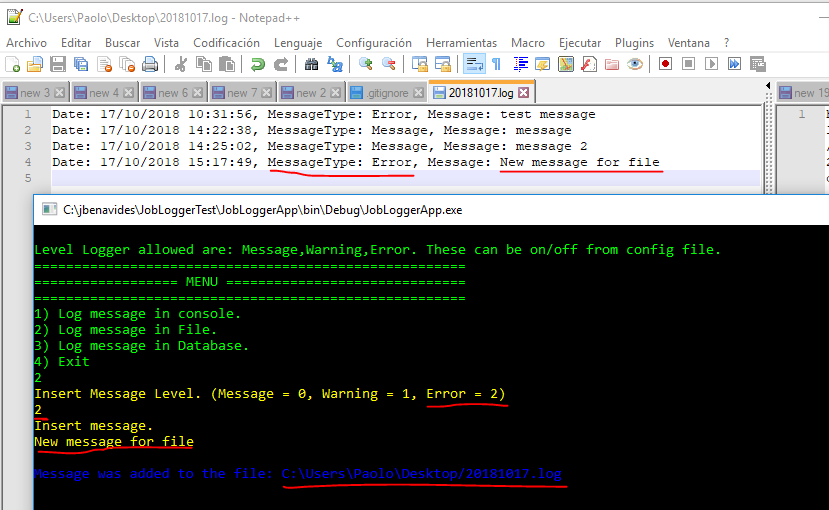
This implementation has a console app that allow us select where we want to log (Console, File, DB). Then we have to input the message type (message, warning, error), The app will check if the message type is allowed if not we are going to have an exception, this configuration can be done in the App.config file.

Here we have some examples:

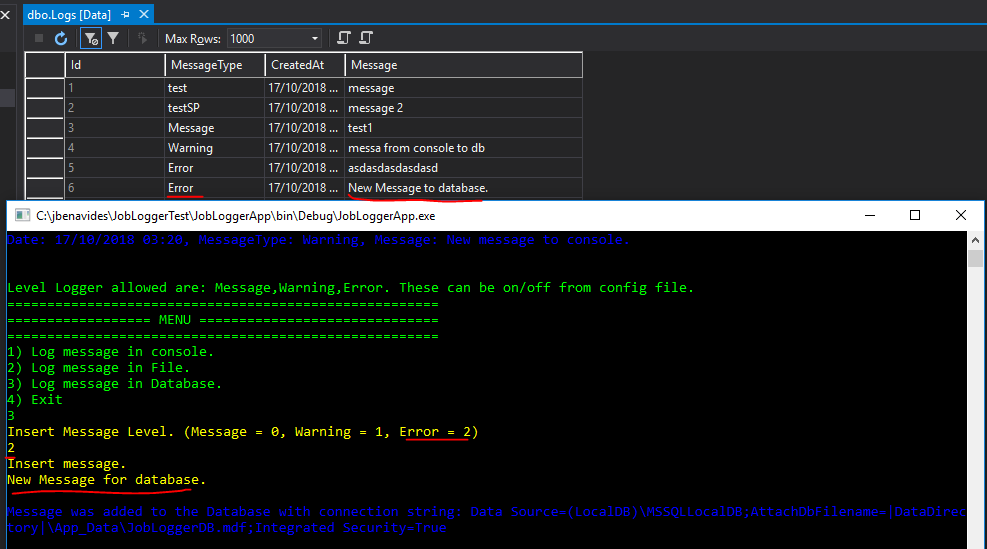
**Console**



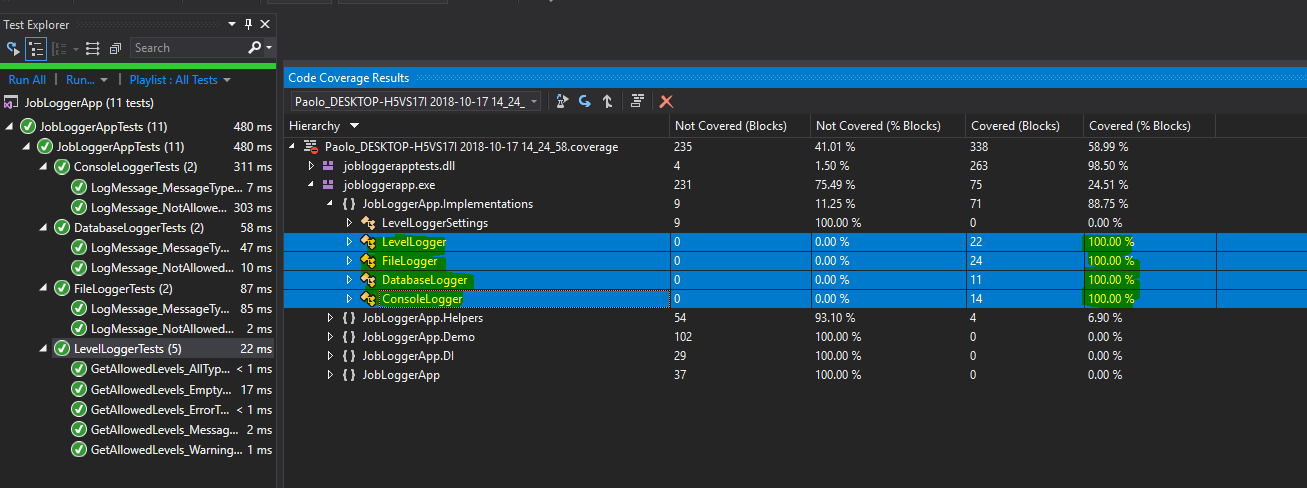
**File**



**Database**



Finally, the unit test and code coverage were made just to the main classes.



Final notes: this exercise could be done using some pattern like abstract factory, but in this case, I decided to do it using the simplest way, using just interfaces and DI.