Design decisions

Folder structure and project structure

Folders are organized in a way that is reasonably easy to find everything – in this case mainly sorted by interface implementation.

Use of interfaces

Interfaces allow for polymorphism and greatly reduce amount of code duplication and make the code easier to test.

Log Entry implementation

LogEntry class uses text, UTC timestamp on message creation and level to give an option to be a bit more verbose. LogEntry ToString() override allows to customize message format across entire logger.

Console Logger implementation

Console doesn't have any async method equivalent, so we delegate this into another thread through usage of Task. We lock the thread to prevent it from being used by another process.

File Logger implementation

File logger requires a path for the file to be created and will append the message to the file rather than overwrite. Used File.AppendAllTextAsync() over StreamWriter.WriteLineAsync(), since the former has native async file I/O. It uses semaphore slim to ensure thread safety while appending text to a file.
In case we run into an error, try catch should be able to write the error to through fallback logger or using already existing console implementation if no argument was specified.

TCP implementation

This is probably most complex implementation as it requires few things before it can be set up properly. It does contain 3 ways of setting up IP End Point through constructor, containing IP address and port to use by the client. It also has a reconnection mechanism that will clean up the connection and will attempt to reconnect again.
In case we run into an error, try catch should be able to write the error to through fallback logger or using already existing console implementation if no argument was specified.

Main Logger Class implementation

Logger contains a list of loggers, which once initialized, is able to register different loggers at the same time and then log the message based upon what loggers were registered.

Then we await the result to complete all tasks. Console log is a bit of an exception as it's a fallback mechanism to log the errors from other loggers through dependency injection, but we can use any other fallback logger if necessary.

Also the logging process is based on a queue, so we can keep adding messages to the logChannel, which will process the messages in the background. Current implementation allows for channels to be processed separately and in order they were added.

The solution is also tracking log messages in order that were added to support graceful shutdown.

FlushAsync is necessary in case if we're trying to wait for the tasks to be completed in a console application, otherwise the code will continue going without waiting for it to finish, prematurely ending execution rather waiting to process entire queue.