Thank you for sending this application, it was very interesting to solve.

I will be happy to discuss the code in greater details however I thought of sharing some design

decision for the application and along with some additional notes.

For running the application, readme page ([link](https://github.com/shantanusrivastv/LogFileParser/blob/master/README.md)) has the required instructions.

# Design

There were various design thoughts like:

1. Having a Json configuration file storing the fields
2. Have configuration to store mapping between common fields and log format type
3. Reading the Fields comment in file to figure out the order and fields then map it.
4. Read the first line of file itself and figure out the fields based on the value (not ideal as some fields are blank denoted by **–** symbol)

The following was chosen

Concrete classes per log file formats having the **exact fields order** and **count** as that of log files and have more **specific data type** like TimeSpan, ushort, instead of generic string. Having specific data types will later help in performing various actions like ranking while displaying the logs.

Since Log File fields can be selected in IIS and even all W3C logs might not have same fields.

We just have to do one-time configuration per log format.

The reason for choosing this approach is following

1. The implementation is very easy for others to understand and we get IntelliSense and compile time support.
2. We want to keep parsing and displaying logs functionality separate. Using this approach this can be easily achieved and we can also harness power of LINQ to generate reports / rank.
3. We can avoid complex logic to determine field /map and also avoid serializing from configuration files.
4. It is very easily extensible both in terms of supporting new Log Formats as well for adding / removing new fields.

## Parsing the Logs

### File Reader and Parser

1. Simple iteration over folder path provide (from config /default) including all subdirectories.
2. ReadAllLinesAsync, with Parallel For loop and ConcurrentBag is used for enhancing both performance and achieving parallelism in the task.
3. Also abstracted away the parsing fields logic based on file format.

For example, IISLogFormat is comma separated and W3C is space separated.

1. Log Parser
2. The design is inspired from .Net base class library of using Parse and TryParse method.
3. If we are sure of file consistency with no invalid data types, we can simply use this Parse.
4. For more complex scenario **TryParse** should be used

* Although expensive, Reflection provide us with the ultimate flexibility and extensibility.
* The code is more generic supporting all file formats.
* We are mapping the log fields with the Log Format class fields and dynamically invoking the TryParse method corresponds to the class type.

For example: **DateTime** call **DateTime.TryParse**, TimeSpan.TryParse and so on.

* This is used instead of creating multiple switch statement for each type as it provides us the greater flexibility of supporting new fields with new data type without any modification following **SOLID principles**.
* Some Performance consideration are also taken like:
  + Using new T (); instead of more expensive Activator.CreateInstance
  + Various check before invoking reflection logic.
  + We can further improve by creating static FieldInfo [] TypeFields field for corresponding LogFileFormat and share it across multiple instances instead of creating it every time. However, have left it to for simplicity and future version as we need to make that thread safe as well.

Is Null/Empty Values  
Those fields which have null, empty values are left unassigned instead of assigning them null.

We can find those fields by default(<type>) check, covered in Unit Test Case.

## Assumptions

* The Time is assumed of UTC hence not used datetime offset or handle different time zones.
* File encoding is assumed to be standard UTF 8

## Testing

3 popular formats for IIS logs is created with corresponding Unit Test Cases

* W3CLogFormat: Original Task
* W3Cv1LogFormat: Older HTTP/1.1 format
* IISLogFormat

(ILogFormat is used just as a label for identifying Log Format Type, it can be further enhanced in future for any additional requirement)

1. Instead of using Mock the concrete implementation is passed, since the application has minimal dependencies and for simplification.
2. Since classes are having dependencies on Interface rather than concrete classes so we can inject Mock dependencies if required.
3. Thorough testing is performed with good code coverage
4. No Unit Testing is created for Client Code, but basic log and exception handling is done.
5. Dependency Injection is implemented as well.

## Display

1. LogViewer is used to display the parsed logs which is again generic implementation supporting all Log File Formats.
2. Sample Client is implemented for W3C Log named W3CLogClient. We can either have similar implementation for other logs or better we can use factory pattern to get instance based on the configuration or other input.
3. Since our type implement IEnumerable we can write complex LINQ statements and display the result as per our requirements.
4. Two similar abstraction for common usage like filtering (where clause) and Group By are implemented which supports wide range of options some of them are shown for reference.
5. We can have similar method or much more generic method accepting Lambda Expression.

Kind Regards,

Kumar Shantanu