**Solution Details**

**Approach**

Requirement has been assessed for technical requirements that can be shared across multiple projects and also analyzed for business specific requirement and identified the following two major use cases.

* **Ability to read and write CSV [Common Technical Requirement]**
  + Read .csv file into strongly typed collections that can be used to process and do required transformations.
  + Ability to write the strongly typed .Net collection as a CSV file
  + Customization of collection to csv and vice versa thru mappers
* **Calculating “GrowthRateInBytesPerHour” for the given dataset and add it as an additional property to the fact data sets. [Business Requirement, Project specific]**

**CsvReadWriter** has been implemented as a separate library in a generic way that it can work with any type of .csv file and .Net <T> type. Highly modular and customizable to work with various shapes of csv schema and type schema. Written with mindset targeting it as a nuget library with various customizations thru constructor overloading so that it can be used for various other scenarios.

**Alternates considered:**

[**FileHelpers**](https://www.filehelpers.net/) and [CsvHelper](https://joshclose.github.io/CsvHelper/) are the two open source alternates identified for this purpose. Decided to not go with them to develop it from scratch specific to this project requirements.

**Project Details**

|  |  |
| --- | --- |
| **Project** | **Purpose Description / Purpose** |
|  | * Read .csv file into strongly typed collections that can be used to process and do required transformations. * Ability to write the strongly typed .Net collection as a CSV file * Customization of collection to csv and vice versa thru mappers   **CsvReadWriteUitlity.Abstractions** – Defines all interfaces and abstract functionality.  **CsvReadWriteUitlity.Implementation** – Implementation of Interfaces/abstractions |
|  | Project for generating the new dataset by transforming/processing the other two given datasets.  This project contains the code and logic to calculate **GrowthRateInBytesPerHour** data for the given files / filestats. |
|  | Unit test project (xUnit testing) |
|  | .Net core Client Console application to run end to end process. |

**Technology Stack:**

|  |  |
| --- | --- |
| **Project / Library/ Approach** | **Technical Stack** |
|  | All the class libraries are developed as **.netstandard 2.0** library to ensure this library works with various different platforms without a need of runtime. |
| Unit Testing | xUnit – Unit testing  FluentAssertions – Assertion framework  NSubstitute - Mock framework |
| Logging | Microsoft.Extension.Logging (.netstandard) |
|  |  |

**Program Execution Instruction**

Run the Client app in the console by issuing the following command. Ensure all the three parameters are passed properly by enclosing them in double quotes (“).

dotnet ServerLogSizeMonitoring.dll -filepath "C:\Temp\Files.csv" -factfilepath "C:\Temp\FileStats.csv" -outputdir "C:\Temp\GrowthInfo"

--------------------------------------------------------------------

All following three parameters are mandatory

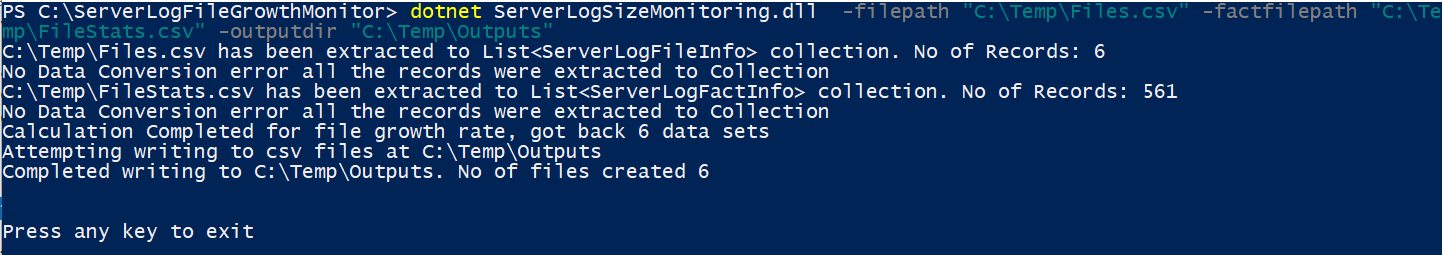
-filepath : Full path of the input file (Files.csv) within double quotes

-factfilepath : Full path of the fact file (FileStats.csv) within double quotes

-outputdir : Output folder where generated fact data sets (1.csv .. 6.csv) are expected to be placed.

Also find the Log folder in the application execution directory to analyze the error/info Logs.

**Screenshot Reference of successful run**



**Improvement Areas/ Technical Debts**

* Unit test coverage can be improved further
* .Net Core client App is not optimized/refactored for better design. Made it simple to handle the basic requirement.
* Dependency Injection at client is not implemented.