# Take Home Test

## Introduction

This problem documentation is not intending to be sufficient to deliver a “production grade product” - it is deliberately unspecific. In a real development situation, you would have the opportunity to ask clarifying questions about requirements - you will not be able to ask such questions in this case. Instead, make assumptions, document and deliver them as part of the solution.

Please use the tools that you are comfortable with.

## System overview

We are implementing a small system for (row-wise) wrangling of text data. The typical use case is taking a delimited data file from a customer and massaging it to fit with a standardized schema by applying a sequence of column transforms.

For example, we could have a target format defined as following (in JVM types):

| Column name | Type |
| --- | --- |
| OrderID | Integer |
| OrderDate | Date |
| ProductId | String |
| ProductName | String (proper cased) |
| Quantity | BigDecimal |
| Unit | String |

Let’s say we get a CSV with the following fields (example provided):

| Column name | String format (pseudo-regex or number parsing format) |
| --- | --- |
| Order Number | d+ |
| Year | YYYY |
| Month | MM |

Day dd

Product Number [A-Z0-9]+

Product Name [A-Z]+

Count #,##0.0#

Extra Col1 --

Extra Col2 --

This source data is missing the `Unit` field, but we know that all products are measured in `kg`.

Transforming from the source to the target could be described with the following steps:

1. Rename `Order Number` → `OrderID` and parse as `Integer`
2. Add a new column, concatenating `Year`, `Month`, `Day` → `OrderDate` and parse as `DateTime`
3. Rename `Product Number` → `ProductId` and parse as `String` 4. Proper case `Product Name` , rename it → `ProductName` and parse as `String` 5. Rename `Count` → `Quantity` and parse as `BigDecimal`
6. Add a new column with the fixed value `"kg"` → `Unit` and parse as `String` 7. Keep only our 6 reference columns

# Requirements

- The transformations should be configurable with an external DSL (like a configuration file)
- The functionality should be implemented as a library, without (significant) external dependencies
- Invalid rows should be collected, with errors describing why they are invalid (logging them is fine for now)
- The data tables can have a very large number of rows

## Out of scope

- Other file formats than CSV
- No need to build a CSV parser unless you really want to - feel free to use a pre-built CSV parser for the basic splitting and escaping

## Deliverables

- Running code and test suite provided through online code repo or in a tar-ball
- Instructions on how to build and run the code with example data
- *Short* architectural overview and technology choices made
- (Basic) documentation, unless it's completely self-documenting (to a fellow software developer)
- List of assumptions or simplifications made
- List of the next steps you would want to do if this were a real project

## What are we looking for?

Try balancing some common sense foresight with the simplest thing that could work. A large chunk of this assignment is very straightforward, but there are also some aspects of both API design and implementation that could be solved in a wide range of ways. Pick a reasonable approach and be prepared to speak to alternatives in the review. This aside, these are the other areas we're scoring during the review:

- A working system
- An extensible system that can perform various transformations
- Easy to use API and easy to understand configuration DSL
- Well structured, readable and maintainable code
- Tests
- Quality of documentation

Good luck, and have fun!

## Submission Format

- Please zip (or tar) up the code and documentation.