Crisp - Back End 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 - Crisp developers are happy to review solutions in any “mainstream” language like Kotlin, Scala, Java, Python, Javascript, Typescript, C# and so on. If you take this opportunity to learn new technologies (and / or language), consider that this might actually reduce the overall quality of your solution and you might spend a lot more time than necessary to implement a good solution.

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 available as a Gist):

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 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 file
- Well structured, readable and maintainable code
- Tests
- Quality of documentation

How much time should I spend?

Short answer:

Spend six hours and turn it in. If your mindset won't let you do that, spend an extra six hours, and *then* turn it in.

Longer answer:

Here at Crisp, we're trying to do what makes sense, instead of what's always been done before. The traditional interview process for a developer usually includes a series of phone screens, an on-line coding test, and culminates with an all-day, on-site interview. The process is costly, intrusive, and fails to replicate what developers actually do on a day-to-day basis.

We're trying to streamline the process, and get at what actually matters: how likely are you to be successful, and thrive, if we welcome you onto our team? Towards that end we've replaced the on-line coding tests, and the on-site interviews with a realistic project that represents something you might actually build at work.

As with any project, you will be presented with the traditional tradeoff between functionality, quality, and cost. We're not looking for perfection, or coverage of every imaginable use case. And we're not looking for you to invest more time in our interview process than you would in a more traditional, on-site interview. Senior developers, using their preferred tools, should be able

to deliver a quality implementation of our take home project, with significant levels of functionality, in about a day.

If that seems insurmountable to you, it does not automatically mean that you are not qualified to work with us. Please, feel free to narrow the implementation scope, or spend a bit more time iterating on your solution - whichever you are more comfortable with. We will ask *you* to set a deadline that you can meet while working at your own pace. It's okay to set that deadline several weeks out, if that's what you will need due to work, life, or family circumstances.

Please do *not* set an extended deadline, intending to invest extraordinary time in this project. We've had candidates be successful having spent as little as four hours, and we've had candidates spend over thirty hours and not advance to the final part of the interview process. We want your time commitment to reflect the investment you would put into any interview process, and not be an undue burden.

Good luck, and have fun!

Submission Format

- Please zip (or tar) up the code and documentation and share it with the recruiter via Dropbox, Google Drive or any other file sharing service you prefer