Functional Programming with Scala

ITI 43 Project

Problem Statement:

A huge retail store wants a rule engine that qualifies orders' transactions to discounts based on a set of ***qualifying rules***. And automatically calculates the proper discount based on some ***calculation rules*** as follows:

QUALIFYING RULES	CALCULATION RULE

less than 30 days remaining for the product to expire (from the day of transaction, i.e. timestamp)	If 29 days remaining -> 1% discount
	if 28 days remaining -> 2% discount
	if 27 days remaining -> 3% discount
	etc …

Cheese and wine products are on sale	cheese -> 10% discount wine -> 5% discount
Products that are sold on 23rd of March have a special discount! (Celebrating the end of java project?)	50% discount !!
bought more than 5 of the same product	6 – 9 units -> 5% discount 10-14 units -> 7% discount More than 15 -> 10% discount

- Transactions that didn't qualify to any discount will have 0% discount.
- Transactions that qualified to more than one discount will get the top 2 and get their average.

- We need the system to be up and running 24 hours. Raw transactions' files will be pushed to the following machine (IP: X.X.X.X) on the following path (……/raw_data). We need the calculations to be done automatically once the file is pushed.

- After ingesting the raw data and calculating the discount please also calculate the final price and load the result in a database table of your choice.

- The raw data needs to be removed from the raw_data directory after successfully writing the processed data in the database.

- It is required to log the engine's events in a log file "rules_engine.log". Please use the following logging format.

```
TIMESTAMP     LOGLEVEL      MESSAGE
```

Technical considerations:

- We can have wrappers of impure functions to interact with the outside world. However, the core logic must be written in a pure functional manner.

In the core functional logic:

- Use only vals, no vars allowed.
- No mutable data structures allowed.
- No loops allowed.
- No Null values.
- Make sure all your functions are pure:
    - Output depends solely on input.
    - Input to the function doesn't get mutated.
    - Has a predictable behavior.

- The code should be well commented.
- The code should be clean, easy to read and self-explainable. (remember that one of the objectives of functional programming is having a readable code)

Greetings,

Please proceed with the new requirements that we agreed on as per our meeting with the head of sales this morning ASAP.

Below is the MOM (Minutes Of Meeting):

1. As per our last week's reports we have very low traffic on the App usage, we need to encourage our clients to use the App more. Please add the following discount rule to our rule-based engine:

QUALIFYING RULES	CALCULATION RULE
Sales that are made through the App will have a special discount	quantity rounded up to the nearest multiple of 5. Ex: if quantity: 1, 2, 3, 4, 5 -> discount 5% if quantity 6, 7, 8, 9, 10 -> discount 10% if quantity 11, 12, 13, 14, 15 -> discount 15% etc …

2. Following the country's long-term goal of going paperless, we want to promote the use of Visa cards instead of cash. Please add the following discount rule to our rule-based engine:

QUALIFYING RULES	CALCULATION RULE
Sales that are made using Visa cards qualify to a minor discount	5%

Thanks,