Ler, Armazenar, Calcular dados

Exercício dividido em 2 partes, a primeira lê e armazena os dados, que serão utilizados na segunda parte, no momento de separar e calcular os dados. **Tecnologias:** Java 8 / Sprint Boot / Maven / Git / JUnit / Mockito. **Diferenciais:** TDD / Explorar os novos recursos do Java 8 (lambdas, streams, etc) / Explorar recursos do Sprint Framework / Organização e estrutura do projeto / Uso do git.

Critérios de avaliação do projeto

- Entregar o projeto em http://github.com ou http://bitbucket.com;
- O arquivo **README.md** deverá conter instruções para compilar e rodar o projeto;
- Iremos rodar o projeto utilizando as instruções no README.md;
- Design e funcionamento da solução;
- Qualidade do código;
- Qualidade dos testes unitários;
- Uso dos recursos da linguagem e do framework;
- Criatividade, tamanho e complexidade da solução;
- Uso de boas práticas e convenções de implementação;
- Versionamento e controle do repositório;
- Possibilidade para evolução da aplicação.

Parte 1: Ler e Armazenar

Construir uma aplicação java8 para ler e armazenar os dados dos 4 arquivos de dados: data_1.json, data_2.json, data_3.json, data_4.json.

Esperamos o processamento mais rápido possível, considere usar paralelismo e pense bem como irá armazenar estes dados. Isso é muito importante pra gente =)

Caso a aplicação seja encerrada inesperadamente (via task manager ou caso dê um erro, etc), ela deverá ser inteligente e, ao subir novamente, ela não deverá processar os dados que já foram armazenados anteriormente.

Deverá validar registros duplicados por arquivo.

Parte 2: Calcular dados

Considerando como exemplo o arquivo **data_1.json**, podemos ver que temos em estoque 270 quantidades do produto EMMS. Podemos ver também que este total de 270 unidades de EMMS foram adquiridas em 4 transações diferentes, cada transação tem sua quantidade (quantity) e seu preço (price). O valor financeiro total do nosso estoque é R$ 1500,19 e o preço médio que gastamos para adquirir este estoque (270 unidades) foi de $5,56 por unidade.

1	file	product	quantity	price	volume
2	data_1.json	EMMS	74	3.75	277.50
3	data_1.json	EMMS	36	5.39	194.04
4	data_1.json	EMMS	99	5.80	574.20
5	data_1.json	EMMS	61	7.45	454.45

sum of quantity	sum of volume	avg price
270	1500.19	5.56

O próximo passo é vender este estoque para lojistas, com o preço mais justo possível entre eles. Considerando 2 lojistas, o resultado final seria:

Lojista 1

	quantity	price	volume
Loja 1	37	3.75	138.75
	18	5.39	97.02
	49	5.80	284.2
	31	7.45	230.95
Qtde	**135**		
Financeiro	**750.92**		
Preço médio	**5.5624**		

Lojista 2

	quantity	price	volume
Loja 2	37	3.75	138.75
	18	5.39	97.02
	50	5.80	290
	30	7.45	223.5
Qtde	**135**		
Financeiro	**749.27**		
Preço médio	**5.5501**		

Repare que

- Ambos os lojistas receberam 135 unidades;
- O preço que cada um pagou ficou próximo do preço médio do estoque. Em alguns casos não será possível cravar o preço médio, porém, deve ser a distribuição mais justa possível;
- A soma dos financeiros e a soma das quantidades batem com o quadro anterior que exibe os totais do arquivo data_1.json.

O que precisamos fazer

Criar um endpoint que faz o cálculo acima, recebendo como parâmetros:

- O **produto** que será distribuído;
- A **quantidade de lojas** que iremos vender nosso estoque.

O exemplo acima foi calculado **apenas com 1 arquivo data_1.json**, considerando o produto **EMMS** para **duas lojas**.

Lembrando que o exercício deverá considerar todos os 4 arquivos JSON durante o cálculo. Esperamos que a Parte 2 utilize os dados armazenados na Parte 1. O programa não poderá ler novamente os arquivos JSON para concluir esta tarefa.