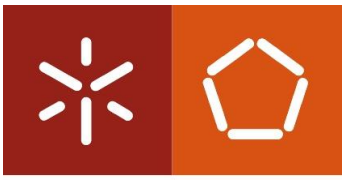

Universidade do Minho

Departamento de Informática

GestVendas

Grupo Nº 1

Paulo Lima (A89983)

Miguel Solino (A86435)

Maria Silva (A83840)

31 de Maio de 2020

Conteúdo

Capítulo 1

Introdução

O objetivo deste projeto é cosntruir um sistema de gestão de vendas, sendo este capaz de armazenar clientes, produtos e informações de vendas relacionadas entre estes. É aplicada uma arquitetura *Model, View, Controller* (MVC) e conhecimentos sobre algoritmos e estruturas de dados.
Como um dos principais objetivos, é necessário garantir encapsulamento dos dados que são armazenados e estes são estruturados de forma a que o sistema seja o mais eficiente possível.
Ao longo deste relatório vamos descrever as nossas abordagens a estes problemas e apresentar alguns testes de performance do nosso projeto final.

Capítulo 2

Problema

São nos fornecidos três ficheiros relativos a uma empresa com três filiais:

- O primeiro contém códigos de produtos.
- O segundo contém códigos de clientes.
- O terceiro contém linhas de vendas entre clientes e produtos e as respetivas informações.

Com base nesses ficheiros é preciso responder a 2 queries estatísticas e 10 queries interactivas:

1. E1. Estatísticas sobre os ficheiros lidos.
2. E2. Números gerais sobre os dados carregados.
3. I1. Códigos de produtos não comprados e totais.
4. I2. Número de vendas realizadas e clientes distintos, num dado mês e numa dada filial.
5. I3. Número de produtos distintos e gastos totais mês a mês para um dado cliente.
6. I4. Determinar mês a mês quantas vezes foi comprado, por quantos clientes e o total faturado de um dado produto.
7. I5. Produtos mais comprados por um dado cliente e a sua quantidade.
8. I6. X produtos mais vendidos todo o ano e o número de clientes que o compraram.
9. I7. Determinar os 3 maiores compradores em cada filial.
10. I8. X clientes que compraram mais produtos diferentes e quantos.
11. I9. X clientes que compraram um dado produto e qual o valor gasto.
12. I10. Calcular a faturação total de um dado produto, mês a mês e filial a filial.

Capítulo 3

Módulos e API

Tendo em conta os objetivos do projeto, e tal como nos foi pedido, decidimos que a melhor arquitetura seria do tipo *Model, View, Controller* (MVC). Um dos critérios que mais nos fez considerar esta escolha foi a modularidade intrínseca desta arquitetura.

3.1 Model

Neste módulo estão presentes todos os algoritmos e dados relacionados com o projeto. É aqui que estão presentes todas os métodos principais para a resolução das queries.

3.1.1 Client

O módulo de Cliente é o responsável pelo tratamento da informação relacionada com o mesmo, incluindo a sua validação.

3.1.2 ClientCatalog

O módulo de Catálogo de Clientes é o responsável pelo armazenando dos clientes lidos e pela pesquisa dos mesmos. Para estrutura principal foi escolhido o Hashmap, presente na biblioteca *java.util.Map*, devido a este ser o mais rápido das estruturas testadas relativamente a pesquisa e inserção.

3.1.3 Product

Semelhante ao módulo de Cliente, o módulo de Produto é o responsável pelo tratamento da informação de um produto individual, incluindo a sua verificação.

3.1.4 ProductCatalog

O módulo de Catálogo de Produtos é o responsável pelo armazenando dos produtos lidos e também pela pesquisa dos mesmos. Como foi dito para o catálogo de Clientes, para estrutura principal foi escolhido o Hashmap, presente na biblioteca *java.util.Map*, devido a este ser o mais rápido das estruturas testadas relativamente a pesquisa e inserção.

3.1.5 Sale

O módulo de Venda, contém os métodos responsáveis pelo parsing de uma linha/string de venda, sendo esta dividida pelos seus campos respetivos na estrutura deste módulo para posteriormente serem usados para atualizar os restantes módulos que estejam relacionados.

3.1.6 Billing

Este módulo, contem todos os métodos responsáveis pela gestão de uma Fatura. Nesta fatura estão presentes os valores totais das vendas de um mês.

3.1.7 BillingProduct

Este módulo, à semelhança do Billing, contém os métodos responsáveis pela gestão de uma Fatura de um produto. Nesta fatura estão presentes os valores totais das vendas desse produto individual.

3.1.8 BillingCatalog

O módulo de Faturas é o principal responsável pela parte da Faturação. Comporta tanto as faturas mensais totais como as faturas mensais de cada produto, sendo estas armazenadas num Hashmap para uma pesquisa mais rápida individualmente.

3.1.9 RelationWithClient

Neste módulo estão presentes todas as informações de todos os clientes que compraram um certo produto. Para uma pesquisa eficiente, a estrutura de armazenamento usada foi um Hashmap.

3.1.10 RelationWithProduct

À semelhança do RelationWithClient, neste estão as informações inversas também em um hash-map, sendo estas de todos os produtos que certo cliente comprou.

3.1.11 Branch

O módulo de filial comporta as ligações que clientes tem com produtos e vice-versa.
Cada filial contém duas Hashtables como estrutura principal, tendo elas as ligações de cada produto com os clientes que o compraram e de cada cliente com os produtos que comprou. As informações presentes nessas ligações são a respetiva faturação.

3.1.12 BranchCatalog

O módulo de catálogo de filiais é capaz de armazenar filiais tendo como entradas iniciais as três filiais pedidas, no entanto é capaz de comportar mais do que essas três caso no futuro exista uma adição de filiais novas.

3.1.13 GestVendasModel

O módulo GestVendasModel é a junção de todos os acima descritos, possuíndo este uma estrutura que contém um Catálogo de Produtos, um Catálogo de Clientes, um Catálogo de Faturas e um Catálogo de Filiais. Este módulo é o principal responsável pelo armazenamento no geral, pelos pedidos do controlador para resolução das queries e carregamento dos dados.

3.2 View

Esta *package* contém as classes de apresentação dos resultados ao utilizador.

3.2.1 GestVendasView

Esta classe possui todos os métodos da view usados pelo controlador para mostrar os resultados ao utilizador.

3.2.2 Pages

A principal função desta classe é dar a possibilidade de se mostrar os resultados através de páginas de forma genérica. Como existem várias queries em que o resultado é uma lista extensa de valores e códigos, esta foi a solução que arranjamos para resolver o problema de mostrar esses resultados extensos.

3.2.3 Table

À semelhança da classe de páginas, esta também foi uma das soluções para a apresentação de resultados de algumas queries. Em algumas o resultado por exemplo é pedido e fornecido separado por meses e filiais, sendo assim a forma que decidimos para mostrar esses resultados é em tabelas.

3.2.4 GestVendasController

Esta classe é a principal responsável pela conexão entre o utilizador e os restantes módulos. Através do Controller é possível o utilizador pedir os resultados das queries pois esta faz o pedido ao Model e com os resultados que recebe de volta, utiliza a View para os apresentar.

3.3 Exceptions

Esta Package contém todas as classes de exceções utilizadas ao longo do projeto. Estas são usadas para assinalar, por exemplo, uma filial ou um cliente inválido passado como argumento.

3.4 Utils

No *package* Utils estão classes que são usadas em várias partes do projeto, e que não fazem necessariamente parte da arquitetura *MVC*.

3.4.1 Constants

A classe Constants contém todas as constantes que foram definidas para o eventual uso ao longo do projeto.

3.4.2 Crono

Esta classe serviu para medir os tempos demorados a correr o projeto, tanto na leitura dos ficheiros, como na execução das queries. Os métodos definidos para este efeito são o *start* e o *stop*. Estes devem ser chamados no inicio e no fim da parte do código a cronometrar, respetivamente. Para além destes dois, implementámos também as funções *getTime* e *getTimeString*, para apresentar o resultado ao utilizador.

Capítulo 4

Testes e Benchmarks

No decorrer do desenvolvimento do projeto, foram feitos vários testes, começando por testes de leitura em que usamos duas formas diferentes, *BufferedReader* e *Files*, e depois testes para decidirmos que estruturas usar no geral, sendo as interfaces testadas *Map*,*Set* e *List*.

4.1 Tempos de execução

Dando uso à classe *Crono* que foi dada pelos professores, realizámos vários benchmarks. Reunindo estes tempos, obtivemos as tabelas C.1, C.2, C.3.
Nos benchmarks realizados, foram feitas experiências com diferentes implementações das Interfaces *Map*, usando *HashMap* e *TreeMap*. Foi concluído que os menores tempos de carregamento das informações foram obtidos com *HashMap*, ficando os tempos de execução das queries quase inalterados. Podemos ver estes resultados nas tabelas C.3.
Para além disto foram testados também os tempos de execução de várias Implementações da Interface *List*, usando para isto *ArrayList* e *Vector*, como podemos ver na tabela C.2. Sendo que não encontrámos diferenças visíveis, acabámos por escolher a implementação mais comum, *ArrayList*.

4.2 Tempos de Leitura

Em relação aos tempos de leitura, foram realizados benchmarks para comparar o desempenho de *Files* com o de *BufferedReader*. Os resultados encontram-se no apêndice B.
Para ler os ficheiros dados, utilizamos o *BufferedReader*, pois, para além de que o *Files* utiliza internamente um *BufferedReader*, vendo os resultados, *Files* mostrou-se mais lento.
Para os testes de leitura e validação de ficheiros, utilizámos o paralelismo intrínseco da JVM8, ao utilizar streams, fazendo com que o tempo de carregamento do ficheiro com 5 milhões de vendas baixasse para quase metade.
Para termos de comparação dos vários ficheiros de vendas, reunimos também os dados em gráficos, apresentados nas figuras B.1 e B.2

Capítulo 5

Conclusão

Para concluir, tendo em conta os requisitos propostos, consideramos que conseguimos implementar tudo o que nos foi pedido de forma eficiente. Todas as queries são respondidas em tempos relativamente baixos, o que para Java é um pouco dificil de se atingir.
Como trabalho futuro gostaríamos de conseguir minimizar os tempos de carregamento de dados e de algumas queries.

Apêndice A

Diagramas de Classes

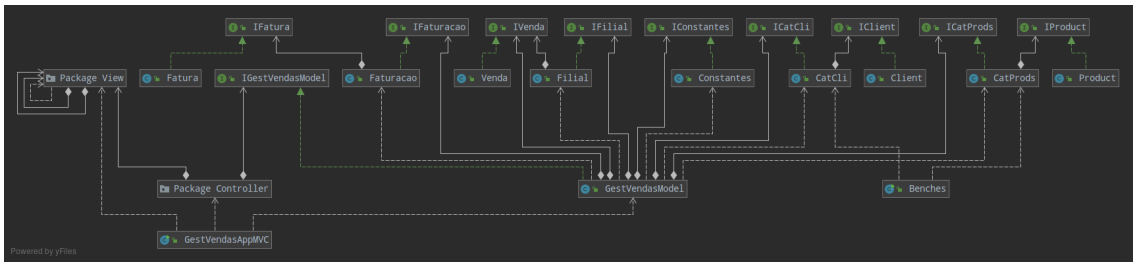

Figura A.1: Diagrama de Classes do Model

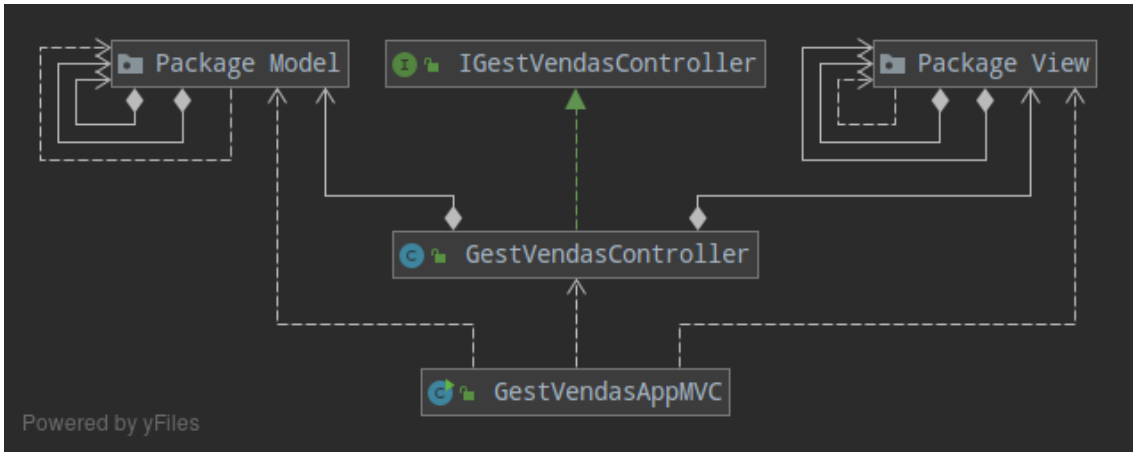

Figura A.2: Diagrama de Classes do Controller

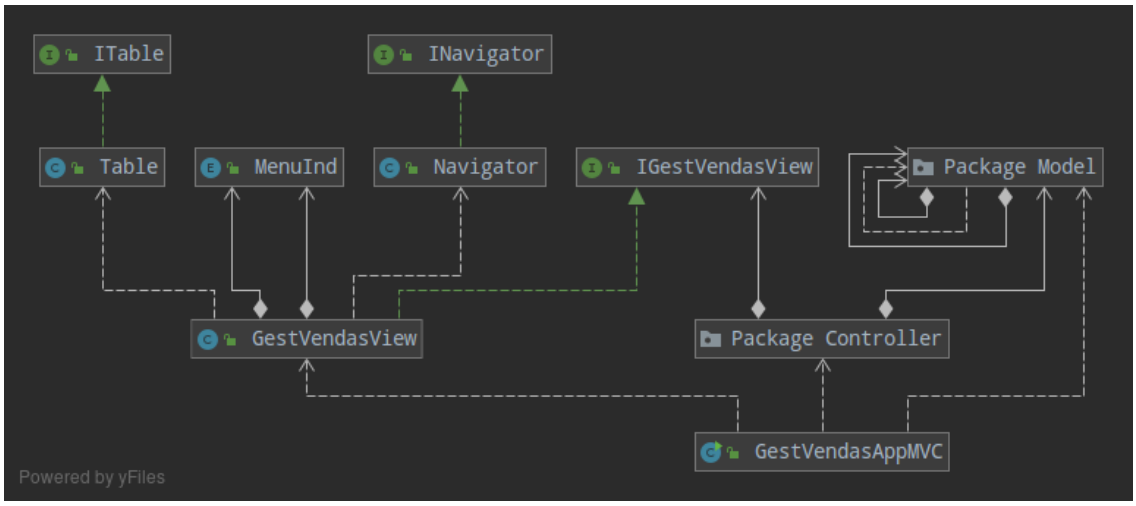

Figura A.3: Diagrama de Classes do View

Apêndice B

Benchmarks BufferedReader vs Files

	1 Milhão	3 Milhões	5 Milhões
Leitura	0.18	0.49	0.78
Parse	0.83	2.05	2.98
Validação	0.82	1.72	2.44
Validação Paralela	0.54	0.97	1.47

Tabela B.1: Tempo (em segundos) de tempos de BufferedReader

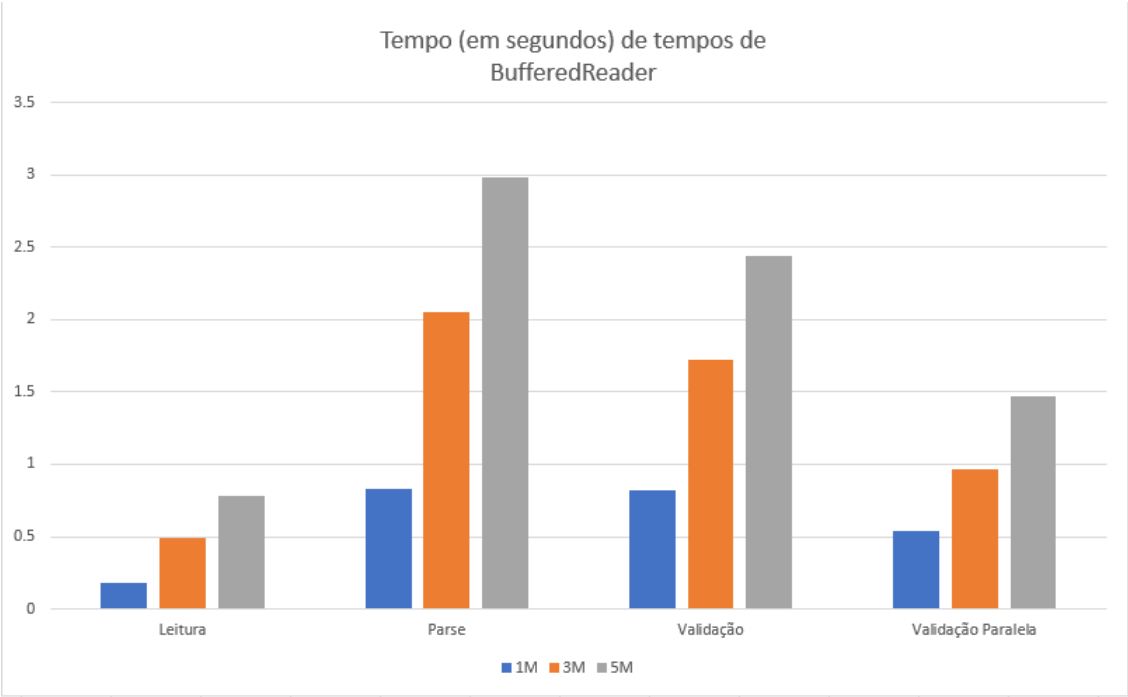

Figura B.1

	1 Milhão	3 Milhões	5 Milhões
Leitura	0.20	0.48	0.84
Parse	0.85	2.06	2.92
Validação	0.81	1.69	2.47
Validação Paralela	0.58	1.09	1.46

Tabela B.2: Tempo (em segundos) de tempos Files

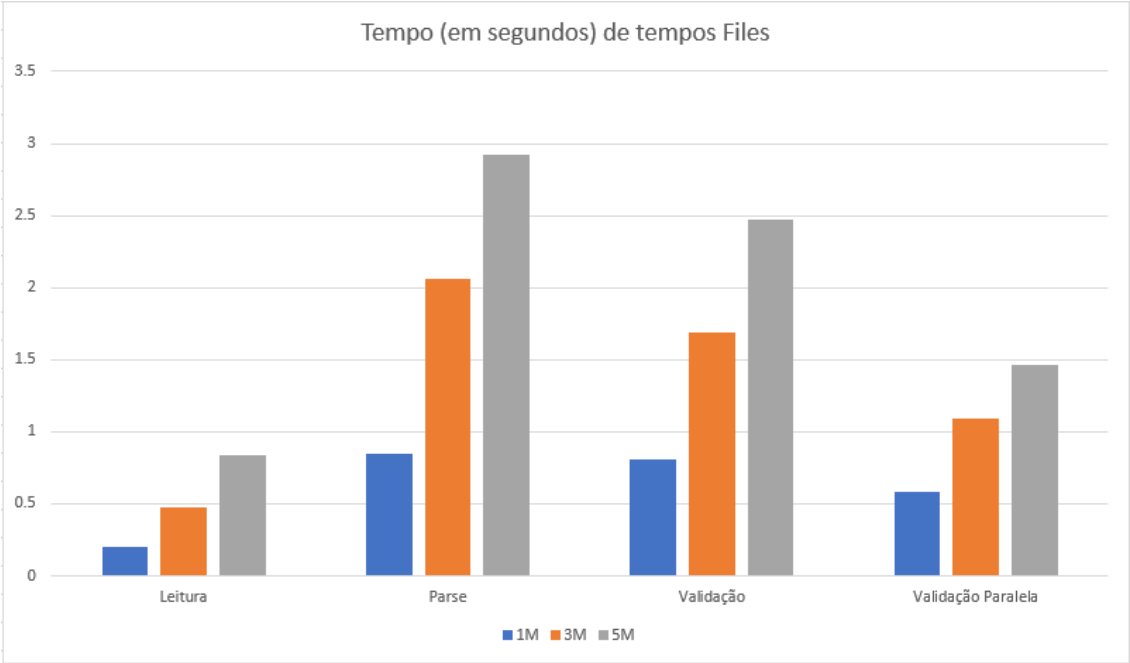

Figura B.2

Apêndice C

Tabela de Tempos de Execução

	1 Milhão	3 Milhões	5 Milhões
Load Time	2.7	8.227	14.122
Query 1	0.1	0.227	0.281
Query 2	0.02	0.066	0.039
Query 3	0.001	0.001	0.001
Query 4	0.002	0.004	0.003
Query 5	0.004	0.002	0.006
Query 6	0.988	1.008	1.154
Query 7	0.105	0.096	0.092
Query 8	0.285	0.894	1.844
Query 9	0.002	0.003	0.003
Query 10	0.035	0.097	0.044

Tabela C.1: Tempo (em segundos) das queries para um dado número de vendas

	1 Milhão	3 Milhões	5 Milhões
Load Time	2.71	8.164	14.123
Query 1	0.147	0.118	0.277
Query 2	0.021	0.05	0.035
Query 3	0.001	0.001	0.001
Query 4	0.002	0.003	0.003
Query 5	0.004	0.002	0.004
Query 6	0.993	1.011	1.162
Query 7	0.102	0.089	0.101
Query 8	0.279	0.902	1.851
Query 9	0.002	0.003	0.003
Query 10	0.0.33	0.0.81	0.0.66

Tabela C.2: Tempo (em ms) das queries para um dado número de vendas, utilizando *Vector* em vez de *ArrayList*

	1 Milhão	3 Milhões	5 Milhões
Load Time	6.41	20.29	33.54
Query 1	0.156	0.259	0.365
Query 2	0.035	0.045	0.042
Query 3	0.001	0.001	0.001
Query 4	0.003	0.004	0.004
Query 5	0.003	0.003	0.003
Query 6	1.012	1.105	1.449
Query 7	0.129	0.092	0.112
Query 8	0.347	0.947	1.817
Query 9	0.001	0.003	0.003
Query 10	0.047	0.062	0.098

Tabela C.3: Tempo (em segundos) das queries para um dado número de vendas, utilizando *TreeMap* em vez de *HashMap*