



UNIVERSIDADE DO MINHO

Mestrado Integrado em Engenharia Informática

Laboratório de Informática III – Projeto Java  
Grupo 64

Ana Filipa Ribeiro Murta (a93284)  
Ana Paula Oliveira Henriques (a93268)  
Augusto César Oliveira Campos (a93320)

Ano Letivo 2020/2021

# Índice

1. Introdução	3
2. Arquitetura do programa	4
2.1. Business e BusinessList . . . . .	4
2.2. Review e ReviewList . . . . .	4
2.3. User e UserList . . . . .	4
2.4. Crono . . . . .	5
2.5. GestReviews . . . . .	5
2.6. LoadLog . . . . .	6
2.7. WriteLog . . . . .	6
2.8. View . . . . .	6
2.9. GestReviewsMVC . . . . .	7
2.10. Testes . . . . .	7
3. Estrutura do projeto	7
4. Testes de performance	8
5. Diagrama de classes	10
6. Conclusão	10

## 1. Introdução

---

Este projeto consistiu no desenvolvimento de uma aplicação *Desktop*, cujo objetivo é a realização de consultas interativas de informações relativas à gestão básica de um sistema de recomendação/classificação de negócios. Visto ser a continuação do projeto C anteriormente realizado nesta unidade curricular, teve como fontes de dados os mesmos ficheiros de texto usados nesse projeto C.

Esta aplicação foi criada em *Java* de forma a pôr em prática os conhecimentos aprendidos na disciplina de *Programação Orientada aos Objetos*. Tal como o projeto desenvolvido previamente, este também é um projeto de programação em larga escala e, por isso, foi necessário recorrer ao uso de estruturas de dados eficientes para armazenar e consultar grandes quantidades de informação (como *HashMap*), garantindo sempre o encapsulamento dos dados.

Neste trabalho prático, embora *Java* já disponibilize estruturas para armazenar enormes quantidades de dados, sem nos termos de preocupar com a alocação e libertação de memória, deparámo-nos com outros desafios, como por exemplo a ordenação de conjuntos segundo um determinado critério ou percorrer os dados dos ficheiros o mais eficazmente possível de forma a poupar no tempo de execução.

**IMPORTANTE:** Por lapso do grupo, não foi criado um Módulo de Estatísticas. Estas encontram-se na classe *GestReviews*, junto das consultas interativas (queries).

## 2. Arquitetura do programa

---

### 2.1. Business e BusinessList

**Classe Business:**                    **private** String businessId;  
(Catálogo de Negócios)        **private** String name;  
   **private** String city;  
   **private** String state;  
   **private** List<String> categories;

**Classe BusinessList:**            **private** List<Business> list;

Esta classe *BusinessList* está encarregue de trabalhar uma lista de negócios.

### 2.2. Review e ReviewList

**Classe Review:**                    **private** String reviewId;  
(Catálogo de Reviews)        **private** String userId;  
   **private** String businessId;  
   **private** float stars;  
   **private** int useful;  
   **private** int funny;  
   **private** int cool;  
   **private** LocalDateTime date;  
   **private** String text;

**Classe ReviewList:**            **private** List<Review> list;

Esta classe *ReviewList* está encarregue de trabalhar uma lista de reviews.

### 2.3. User e UserList

**Classe User:**                    **private** String userId;  
(Catálogo de Users)        **private** String name;  
   **private** List<String> friends;

**Classe UserList:**            **private** List<User> list;

Esta classe *UserList* está encarregue de trabalhar uma lista de users.

O objetivo em criar estas classes *BusinessList*, *ReviewList* e *UserList* foi para se poder armazenar de forma eficaz um grande número de objetos dos tipos *Business*, *Review* e *User*, respetivamente. Por exemplo, para guardar todas as reviews lidas de um ficheiro numa *ReviewList* ou para guardar todos negócios de um user num *BusinessList*.

## 2.4. Crono

Esta classe foi facultada pelos docentes desta unidade curricular e é usada por outra classe para calcular os tempos de execução de cada query. Recorre, por isso, ao método `System.nanoTime()` para medir as diferenças de tempo em nanosegundos que são convertidas para segundos e milissegundos.

## 2.5. GestReviews

```
private BusinessList bus;  
private ReviewList rev;  
private UserList user;
```

A classe *GestReviews* tem como variáveis de instâncias as listas de negócios, de reviews e de users já preenchidas a partir dos ficheiros facultados e responde a todas as queries, tanto as estatísticas como as consultas interativas, com estas variáveis de instância. Esta classe permite, ainda, que armazenemos o seu estado atual através da *ObjectStreams* e que guardemos esse estado num ficheiro.

### Estratégia da Estatística 1:

Decidiu-se apresentar, para cada ficheiro, todos os dados que dizem respeito a esse ficheiro. Por exemplo:

**Nome do ficheiro:** reviews.csv  
Número de reviews errados: 0  
Número de reviews com 0 impacto: 32

**Nome do ficheiro:** business.csv  
Número total de negócios: 2051  
Número de negócios avaliados: 1997  
Número de negócios não avaliados: 54

**Nome do ficheiro:** users.csv  
Número total de users: 1894  
Número de users que fizeram reviews: 1802  
Número de users que nada avaliaram: 92

É importante denotar, no entanto, que, por lapso do grupo, não foi estabelecida uma correta ligação entre o *Model* e a *View* ao serem mostrados os resultados desta estatística ao usuário visto que são feitos prints neste método.

### Estratégia da Estatística 2:

Para responder a esta estatística, decidiu-se preencher três arrays, recebidos como argumentos: um que guarda o número de reviews por mês, outro que guarda a média da classificação de reviews por mês e outro que guarda o número de users distintos que avaliaram por mês.

Cada índice dos arrays representa, portanto, um mês:

Jan	Fev	Mar	Abr	Maio	Jun	Jul	Ago	Set	Out	Nov	Dec
0	1	2	3	4	5	6	7	8	9	10	11

A estratégia adotada para determinar o número de utilizadores distintos que avaliaram por mês foi criar um *HashSet* onde, à medida que se percorria a lista de reviews, se o id do utilizador que fez uma review ainda não existisse no *HashSet*, seria guardado nesse conjunto e, posteriormente, seria incrementado o valor do respetivo array na posição do mês em que se encontra.

### Estratégias das Consultas Interativas:

Recorreu-se a conjuntos como *HashSet* para verificar a não repetição de dados, *TreeSet* para ordenar listas e *Map* para associar um determinado value a uma key. Assim, sempre que se quisesse obter a lista de reviews de um user específico, por exemplo, bastava recorrer ao *Map* onde estes valores estavam armazenados.

Para além do recurso a estes conjuntos, também se recorreu a *Comparators* para ordenar *Map's* segundo um determinado critério (ou até mais do que um). Isto facilitou imenso, por exemplo, a ordenação decrescente de um *Map* segundo o número de negócios diferentes que um user avaliou, sem quebrar a associação key-value previamente estabelecida.

## 2.6. LoadLog

A classe *LoadLog* é responsável por carregar objetos do tipo *BusinessList*, *ReviewList* e *UserList* com o conteúdo lido dos ficheiros *business\_full.csv*, *reviews\_1M.csv* e *users\_full.csv*, respetivamente. Para isso, através do polimorfismo, temos métodos *parse* adaptados a cada um dos ficheiros de maneira a verificar a validação dos dados e temos, ainda, um método *getFichDefault()* que devolve uma string com os nomes de cada um dos ficheiros a serem lidos.

## 2.7. WriteLog

A classe *WriteLog* é responsável por gravar num ficheiro de objetos o estado do programa (a estrutura de dados).

## 2.8. View

A classe *View* trabalha a parte visual do programa, ou seja, comunica com o utilizador, mostrando-lhe os menus do programa, os resultados de cada uma das queries e os input prompts.

## 2.9. GestReviewsMVC

A classe *GestReviewsMVC* é a classe principal visto que controla o fluxo de todo o programa. A partir do input do usuário, esta classe executa a query escolhida pelo mesmo e controla, também, os ficheiros que serão carregados no programa. Para além disto, esta classe possibilita ao utilizador a opção de correr o método `runTestes()` da classe *Testes*.

## 2.10. Testes

Tal como o nome sugere, esta classe é responsável por realizar os testes de performance do programa. Assim, através da classe *Crono*, são calculados os tempos de execução de cada query, bem como a memória usada por cada uma.

## 3. Estrutura do projeto

---

Tendo como estrutura *Model-View-Controller*, o nosso projeto encontra-se organizado do seguinte modo:

Model

A camada de dados é constituída pelos catálogos *Business*, *Review* e *User*, em conjunto com as classes *BusinessList*, *ReviewList* e *UserList*, e pela classe *GestReview* que trabalha todas as queries requeridas.

View

A camada de interação com o utilizador, estando encarregue da apresentação do programa, é apenas constituída pela classe *View*.

Controller

A camada de controlo do fluxo do programa, que interage com as outras duas camadas para que estas não tenham de comunicar diretamente uma com a outra, é constituída pelas classes *GestReviewsMVC* e *Testes*.

Portanto, é mostrado ao usuário os menus da aplicação e, perante a escolha do mesmo, o *Controller* está encarregue de enviar o pedido ao *Model* (por exemplo, efetuar os testes de performance). O *Model* concretiza o pedido e envia a resposta ao *Controller*, que pede à *View* para mostrar a resposta ao utilizador.

## 4. Testes de performance

---

Para contextualização dos testes, segue-se as especificações do computador em que os testes foram executados. Vale a pena também referir que foram executados na distribuição Mint Cinnamon, baseada na distribuição Ubuntu que, por sua vez, é baseada em Debian.

### Processador

① CPU	AMD Ryzen 7 4800H
① Família de Processador	AMD Ryzen 7
① Velocidade Processador	2.9 GHz (Turbo: 4.2GHz)
① Arquitetura CPU	Zen 2
① Número de Núcleos Core	Octa Core

### Memória e Armazenamento

① Memória RAM	16 GB
① Armazenamento	512 GB SSD PCIe
① Tipo de Armazenamento	SSD PCIe

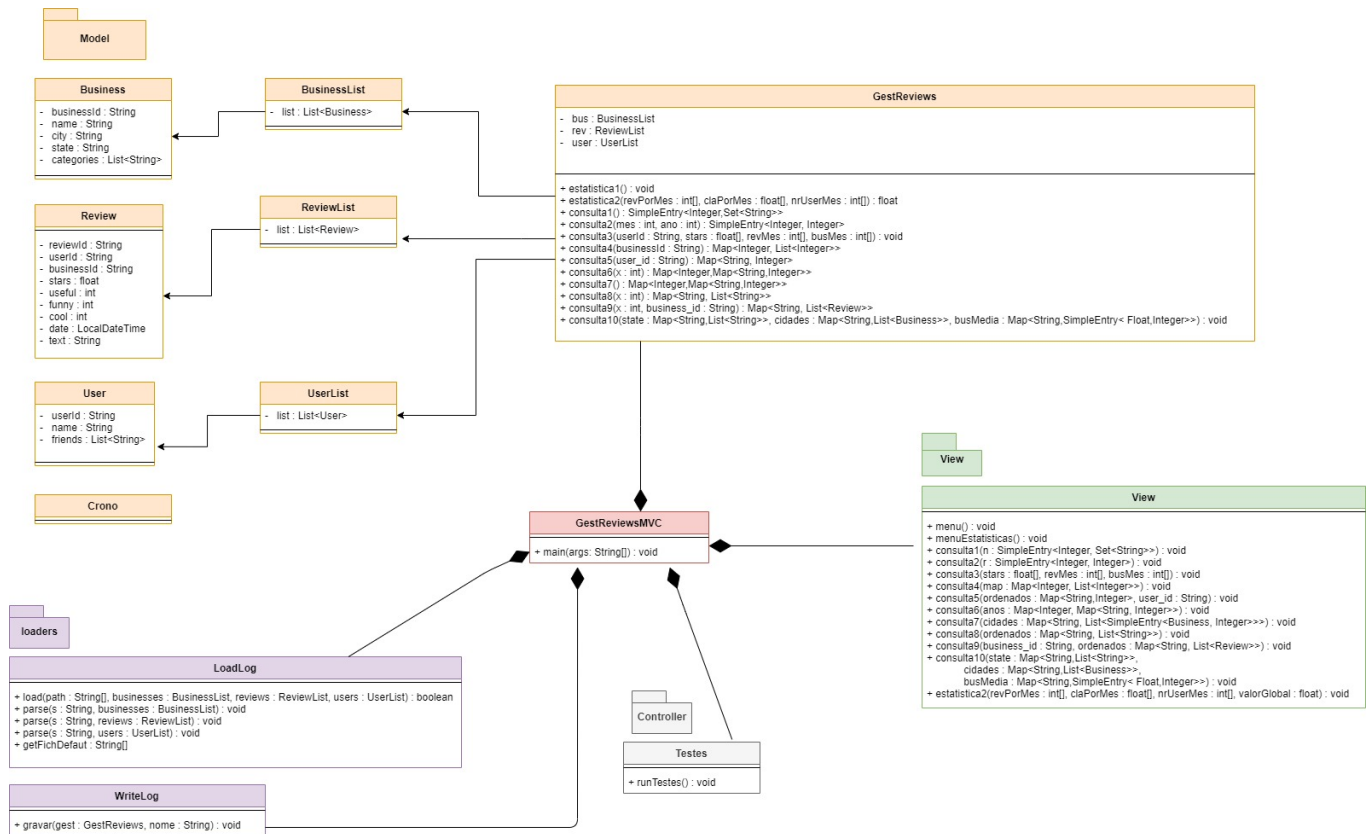
### Placa Gráfica

① Gráfica	NVIDIA Geforce RTX 2060
① Tipo de Placa Gráfica	NVIDIA GeForce RTX
① Memória GPU	6 GB



Funcionalidade	Tempo (segundos)	Memoria (MB)
Estatística 1	1.014	23.498
Estatística 2	0.061	23.498
Consulta 1	0.274	30.000
Consulta 2	0.085	30.000
Consulta 3	0.753	30.498
Consulta 4	0.498	30.498
Consulta 5	0.609	30.498
Consulta 6	0.659	44.690
Consulta 7	0.411	98.192
Consulta 8	0.664	63.603
Consulta 9	0.059	63.851
Consulta 10	28.095	149.851

## 5. Diagrama de classes



## 6. Conclusão

Apesar de termos respondido a todos os requisitos propostos, temos consciência que o trabalho prático podia estar melhor estruturado e melhor conseguido. De facto, acreditamos que a nossa aplicação responde a todas as queries eficientemente e em tempos de execução aceitáveis. No entanto, ao longo do projeto, escaparam-nos alguns erros que poderiam ter sido evitados se tivéssemos tido mais atenção e cuidado.

Por outro lado, procurámos melhorar aspetos negativos que foram apontados ao projeto C previamente entregueado, como por exemplo construir catálogos para cada um dos objetos *Business*, *Review* e *User*. Daí, acreditamos que o nosso projeto Java está melhor estruturado e mais simples de entender que o projeto C.