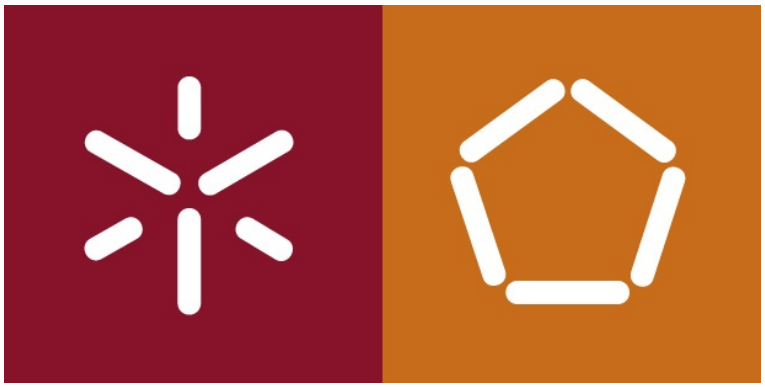

# Relatório do Projeto de Laboratórios de Informática III

2.º ano de MIEInf

**Artur Ribeiro - A82516**
**Davide Matos - A80970**
**Francisco Freitas - A81580**
**Inês Alves - A81368**

**Departamento de Engenharia Informática**
**Junho de 2018**

# Conteúdo

# 1 Introdução

No âmbito da Unidade Curricular Laboratórios de Informática III, do Mestrado Integrado em Engenharia Informática da Universidade do Minho, este projeto tem como objetivo o desenvolvimento de um sistema capaz de processar ficheiros XML que armazenam as várias informações utilizadas pelo *Stack Overflow*.

Uma vez processada esta informação, pretendia-se que fosse possível executar um conjunto de questões específico de forma eficiente. Esta aplicação foi desenvolvida pela segunda vez pelo grupo, desta vez na linguagem de programação Java, conforme solicitado no enunciado do projeto.

Devido à grande quantidade de dados, este projeto tornou-se um desafio, uma vez que, ao terem que passar no nosso programa milhões de dados, tornar-se-ia fundamental escolher bem as esturuturas de dados a utilizar de modo a que as questões propostas fossem respondidas de forma eficiente.

- O que é o Stack Overflow?

O *Stack Overflow* é, atualmente, uma das comunidades de perguntas e respostas mais utilizadas por *developers* em todo o mundo. Nesta plataforma, qualquer utilizador pode colocar questões que serão depois respondidaspor outros utilizadores. Estas respostas estão sujeitas a um sistema de votações que tende a favorecer as melhores respostas dado que as respostas são apresentadas por ordem decrescente do número de votos.

Cada utilizador acumula pontos de reputação que são ganhos sempre que uma das suas respostas é favoravelmente votada *(voted up)* e perdidos sempre que uma das suas respostas é votada desfavoravelmente *(voted down)*.

Segundo informação oficial, a plataforma tem cerca de 8.4 milhões de utilizadores e já se colocaram cerca de 15 milhões de questões, às quais se responderam 24 milhões de vezes.

A informação resultante da utilização da plataforma é extremamente útil e valiosa, não só devido ao conteúdo das perguntas e respostas mas também aos metadados originados a partir dos mesmos. Através deles é possível inferir, por exemplo, quais são os utilizadores mais ativos ou quais são os temas (identificados por tags) mais comuns.

A análise destes metadados pode, no entanto, ser um processo bastante demorado e custoso devido ao grande volume de dados com os quais se tem de lidar e às operações necessárias para cruzar as diferentes informações disponíveis.

## 2 Estruturas de dados usadas

É importante referir, neste ponto, a diferença entre este projeto em *Java* para o projeto realizado anteriormente em C.

Consideramos que, nesta linguagem, o *parser* foi mais fácil de implementar uma vez que a informação disponível nos motores de busca por todos nós conhecidos era muito maior. Neste projeto guardar a informação desejada também foi relativamente mais simples uma vez que não foi necessário lidar com problemas de memória como é costume em C. Isto é, implementando, por exemplo, um *TreeMap*, não tivemos que nos preocupar com alocar memória, sendo apenas necessário inserir os elementos e posteriormente, caso necessário, ordená-los.

Quanto às estruturas de dados utilizadas, podemos afirmar que foi necessária uma primeira análise mais detalhada sobre os ficheiros que continham a informação base à realização do nosso projeto. Depois de feita esta análise, uma leitura atenta das *queries* a que queriamos dar resposta foi também importante.

Posto isto, podemos destacar três entidades "principais":

- *Tags*
- *Posts*
- *Users*

Para cada uma destas entidades foi, como era de esperar, criada uma classe: *Tags*, *Posts* e *Users*.

No que toca à organização de toda a informação foi utilizada, principalmente, a interface *Map* da, já referida, linguagem *Java*.

É de realçar que na versão anterior, na linguagem de programação C, cada *User* tinha a lista dos seus *Posts*, enquanto que nesta versão utilizamos um *HashMap* que relaciona o ID de cada *User* à lista dos seus *Posts*.

Resumindo, implementamos quatro *TreeMap*:

1. Map <Long,Posts>posts;

   Relaciona o ID de um *Post* a esse mesmo *Post*.

2. Map <Long,Users>users;

   Relaciona o ID de um *User* a esse mesmo *User*.

3. Map <Long,List<Posts>>postsUser;

   Relaciona o ID de um *User* à lista de *Posts* desse mesmo *User*.

4. Map<LocalDate,List<Posts>>dataposts;

   Relaciona uma dada data à lista de *Posts* nessa mesma data.

E um *HashMap*:

- Map<Long,Tags>tags;

  Relaciona uma dada *Tag* ao ID dessa mesma *Tag*.

Para além destas estruturas essenciais decidimos ainda implementar algumas estruturas auxiliares para garantir uma rápida resposta às *queries*.

# 3 Modularização funcional e Abstração de dados

- **Encapsulamento**

O encapsulamento permite-nos que só exista uma forma de aceder aos nossos dados, através *"getters"*, e uma forma de os alterar, através de *"setters"*.

Estes métodos permitem devolver sempre uma cópia da informação contida na nossa estrutura, evitando que dados específicos sejam acedidos ou usados diretamente, o que traz uma maior segurança, uma vez que evita que um programa se torne tão interdependente que uma pequena mudança tenha grandes efeitos colaterais.

```
public long getID() {
    return this.ID;
}
```

Figura 1: Função *getID*

# 4 Estratégias seguidas em cada uma das interrogações

- *Query 1*

  A estratégia seguida para responder a esta interrogação consiste numa simples "busca"de informação de um dado *Post*. Dado o ID de um *Post*, retorna a informação relativa a esse *Post*, ou seja, o título do *Post* e o nome de utilizador do autor do mesmo.

- *Query 2*

  Esta *Query*, cujo obtem os N utilizadores com mais atividade, ou seja, com maior número de posts de sempre, sendo consideradas, para este feito, tanto perguntas como respostas. Para este efeito, foi criada uma lista com todos os *Users* a partir do *Map* que relaciona o ID de um *User* a esse mesmo, já referido anteriormente. Esta lista é ordenada através da classe *ComparatorUPosts* que ordena por ordem decrescente a lista em função do número de *Posts* dos *Users*. Por fim, o método para dar resposta à *Query* retorna os N primeiros dessa lista.

- *Query 3*

  Para dar resposta a esta *Query* é criada uma lista de *Posts* que estão compreendidas no intervalo de tempo dado. Feita esta lista é feita uma separação dos *Posts*, sendo as perguntas e as respostas distinguidas umas das outras. Esta distinção é aproveitada no fim para se retornar um *Pair* com todas as perguntas e todas as respostas.

- *Query 4*

  Nesta *Query* foi criada uma lista de *Posts* que continham a *Tag* dada no intervalo de tempo dado. Para ordenar a lista pelo critério pedido (por cronologia inversa) foi implementada uma nova classe: *ComparatorPostsDataInv*. Como resposta é retornada essa lista já com a aplicação desta nova classe.

- *Query 5*

  Como estratégia para resolver esta *Query*, dado um ID, fomos ver qual era o *User* a que pertencia esse ID. Foi também criada uma lista com todos os *Posts* desse autor, sendo posteriormente ordenada por cronologia inversa. Após esta ordenação, outra lista é criada para guardar os IDS desses *Posts*, sendo apenas selecionados os 10 primeiros *Posts* da lista anterior, ou seja, os 10 últimos *Posts* deste autor, tal como pedido no enunciado, uma vez que está ordenada por ordem cronológica inversa.

- *Query 6*

  Para esta *Query* foi criada uma lista de *Posts* que contém todas as respostas compreendidas no intervalo de datas dado. A esta lista é aplicada a classe *ComparatorPostsScore* que compara 2 *Posts* em função do seu *score*.

  São retornadas as N primeiras respostas desta lista ordenada.

- *Query 7*

  Para esta *Query*, primeiramente são filtradas as perguntas que se encontram no intervalo de datas passado. Após esta filtragem, são ordenadas por ordem decrescente do AnswerCount, sendo depois devolvidos os IDs das N primeiras perguntas da lista resultante.

- *Query 8*

  Nesta *Query* são guardados todos os *Posts* cujo título contém a palavra dada numa lista que é posteriormente ordenada através da aplicação da classe *ComparatorPostsDataInv* por cronologia inversa.

  Feita esta ordenação são devolvidos os N primeiros elementos desta lista.

- *Query 9*

  Como estratégia para resolver esta *Query*, foram guardadas as perguntas em que cada um dos utilizadores dados participou (seja via pergunta ou via resposta) e são guardadas numa lista as que forem comuns aos 2 utilizadores. Esta lista é novamente ordenada por cronologia inversa, sendo os N primeiros IDs desta lista a resposta a esta *Query*.

- *Query 10*

  Para dar resposta a esta *Query*, através de uma procura simples da pergunta desejada a partir do seu ID, é calculada a melhor resposta a essa pergunta através dos cálculos indicados, sendo esta mesma resposta a resposta à *Query*.

- *Query 11*

  Nesta *Query* são guardados os N utilizadores com maior reputação no intervalo de tempo dado, ou seja, os utilizadores com mais *Posts* naquele intervalo de tempo. Por fim, novamente no memso intervalo e guarda-se as *Tags* que aqueles *Users* usaram.

  Por fim são devolvidos os IDs destas *Tags* ordenados por ordem decrescente do número de vezes que foram usadas.

# 5 Conclusão

Ao longo do projeto, as dificuldades encontradas foram bastantes e diversificadas. Devido a estas dificuldades é necessário também referir que os *commits* apresentados no *GitHub* não correspondem à realidade, uma vez que nem todos os elementos do grupo tinham equipamento adequado à realização do trabalho. No entanto, de modo a que nenhum dos elementos saísse prejudicado, foram feitas inúmeras sessões de trabalho em que todos discutiram e partilharam ideias, tornando a realização deste projeto mais clara e proveitosa para todos.

A realização deste projeto, para além de permitir o desenvolvimento das capacidades de raciocínio e programação do grupo, serviu como primeiro contacto no que toca ao uso de bibliotecas deste género, análise de grandes quantidades de informação e manuseamento de trabalho e implementações já existentes.

De facto, o maior obstáculo deste projeto, além da programação em larga escala, ao contrário do que acontecia nas Unidades Curriculares anteriores, foi o estudo de diversas bibliotecas, e a obtenção de uma linha de raciocínio que tivesse como prioridade a eficiência de resposta, que seria um dos maiores objetivos.

Apesar de todas estas dificuldades, consideramos que a realização deste projeto, nesta linguagem, foi mais simples do que na linguagem de programação C (já realizada e discutida anteriormente com sucesso).