Guião III

Laboratórios de Informática III

2021/2022

Objetivos

- Consolidação dos conceitos de **modularidade** e **encapsulamento** aplicados no guião anterior;
- Suporte a diferentes mecanismos de interação entre programa e utilizador;
- Manipulação e otimização de consulta de grande volume de dados;
- Testes funcionais e avaliação de desempenho.

Realização e avaliação do trabalho desenvolvido

O desenvolvimento deste projeto deve ser realizado colaborativamente com o auxílio de *Git* e *GitHub*. Os docentes serão membros integrantes da equipa de desenvolvimento de cada trabalho e irão acompanhar semanalmente a evolução dos projetos. A entrega do trabalho será efetuada através desta plataforma e os elementos do grupo serão avaliados individualmente de acordo com a sua contribuição no repositório. Serão fornecidas algumas regras que os grupos deverão seguir para manter e estruturar o repositório de forma a que o processo de avaliação e de execução dos trabalhos possa ser uniforme entre os grupos e que possa ser efetuada de forma automática.

- O trabalho desenvolvido por cada grupo será avaliado com base no conteúdo da pasta “guiao-3” do repositório GitHub, à data de 23 de Janeiro (23:59);

- O relatório do trabalho (máximo de 10 páginas de conteúdo, ou seja, excluindo capas ou anexos) terá de ser disponibilizado na raiz da pasta “guiao-3” até dia 23 de Janeiro (23:59), em formato PDF, e o ficheiro correspondente terá de ter o nome “relatorio-3.pdf”. O relatório deverá centrar-se na resolução dos exercícios propostos, identificando as estratégias seguidas e eventuais limitações, bem como a medição de aspectos de desempenho, tais como o custo computacional e de armazenamento das estruturas de dados;

- Esta fase do trabalho a realizar na unidade curricular de Laboratórios de Informática III terá um peso de 40% na avaliação final. A avaliação desta fase será composta por 80% do exercício + 20% do relatório;

- O trabalho terá de ser desenvolvido por todos os elementos do grupo de trabalho e todos deverão registar as suas contribuições individuais no respectivo repositório;

- O projeto terá de gerar o necessário ficheiro executável com base na preparação de um ficheiro “Makefile” (na raiz da pasta “guiao-3”) e por invocação do comando “make”. Da mesma forma, deverá limpar todos os ficheiros desnecessários ao projeto através da execução do comando “make clean”;

- O executável deverá assumir a existência de ficheiros de entrada com nomes “users-g3.csv”, “commits-g3.csv” e “repos-g3.csv” dentro de uma sub-pasta “entrada” da pasta raiz da pasta “guiao-3”;

- Os ficheiros de entrada iniciais poderão conter registos inválidos – ver guião 1 – pelo que o programa deverá garantir que as queries são executadas sobre um conjunto de dados coerente;

- O trabalho realizado por cada um dos grupos será avaliado em sessões de discussão com a equipa docente. Estas sessões decorrerão entre os dias 24 e 26 de Janeiro.

Exercícios

Mecanismo de interação

Para além do mecanismo de interação desenvolvido no guião anterior, requisita-se agora o desenvolvimento de um mecanismo de interação alternativo através de um menu interativo que torne a utilização do programa mais intuitiva e visualmente apelativa para um utilizador padrão. Desta forma, o programa deverá disponibilizar este menu sempre que não receba um ficheiro de comandos como argumento. Caso o programa tenha sido invocado desta forma (sem argumentos), este deverá processar os ficheiros de dados padrão e apresentar ao utilizador um menu onde liste as diferentes funcionalidades que o programa disponibiliza e solicitar uma opção ao utilizador. Após a escolha por parte do utilizador, o programa deverá retornar os resultados da opção escolhida para o *stdout* num formato visualmente apelativo e que facilite a consulta e navegação pelos resultados obtidos. Desta forma, pretende-se que os resultados sejam retornados num sub-menu que pagine os resultados e permita a navegação pelas diferentes páginas. A dimensão das páginas poderá ser fixa ou determinada consoante a dimensão da janela da linha de comandos.

```
$ ./guiao-3

——------------------------------------------------------------------
| 1  |  Quantidade de bots, organizações e utilizadores   |
——------------------------------------------------------------------
| 2  |  Número médio de colaboradores por repositório    |
——------------------------------------------------------------------
…
——------------------------------------------------------------------
| 10 |      Top N de utilizadores por cada repositório        |
```

```
——------------------------------------------------------------------

Insira opção:
10
Insira número de utilizadores:
3

 ——————————— Repositório 1234 ———————————
|ID1|Login1|Commit_msg_size1|1234|
|ID3|Login2|Commit_msg_size2|1234|
|ID3|Login3|Commit_msg_size3|1234|
 ——————————— Repositório 1235 ———————————
|ID4|Login4|Commit_msg_size4|1235|
|ID5|Login5|Commit_msg_size5|1235|
|ID6|Login6|Commit_msg_size6|1235|
--------------------------------
-------- Página 1 de 255 --------
P     -> Próxima
A     -> Anterior
S <N> -> Saltar para página

Insira opção:
```

Testes funcionais e de desempenho

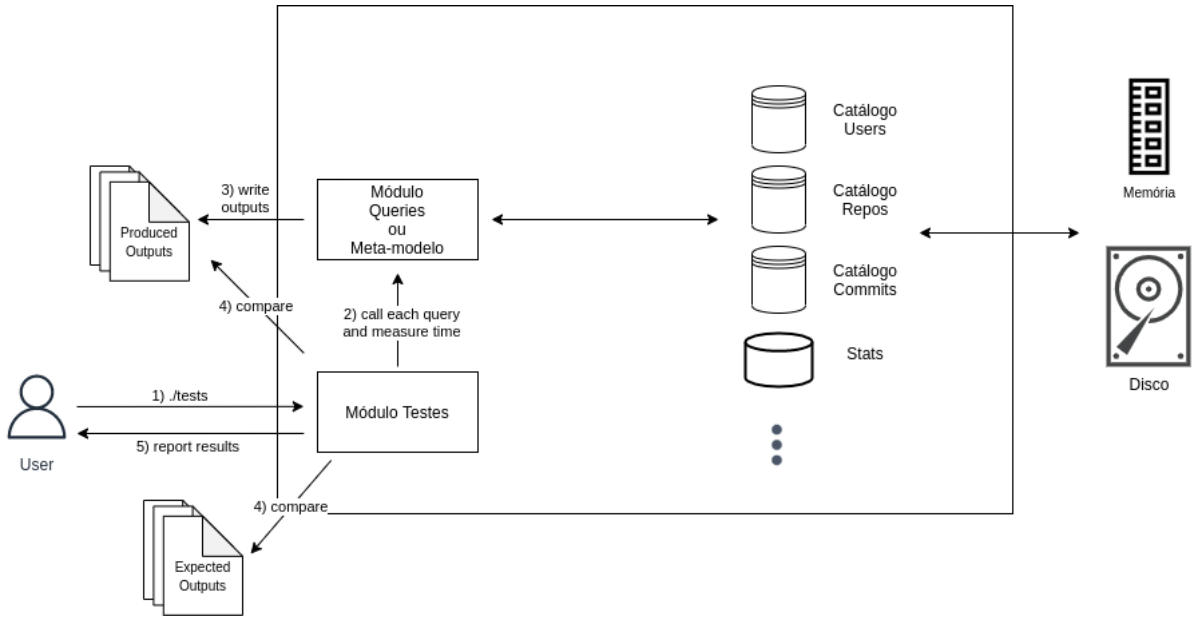
Neste novo exercício, pretende-se que sejam desenvolvidos testes que validem e avaliem o funcionamento do programa desenvolvido. Desta forma, pretende-se que sejam desenvolvidos testes que avaliem o funcionamento de cada *query* requisitada no guião anterior. Para cada uma destas funcionalidades, deverá ser desenvolvido um teste que avalie se a query é executada em tempo útil (consideremos como tempo útil um tempo de execução inferior a **5 segundos**) e que o seu resultado é correto.

```
test_top_users_msg_size_per_repo() {
    query10(); //write result to file
}

main() {
    start_time = time_now();
    test_top_users_msg_size_per_repo();
    time_passed = time_now() - start_time;

    compare(produced_file, expected_file);
}
```

Pseudo-código de um módulo de testes que avalia e valida a query 10



Exemplo **ilustrativo** de um possível fluxo de execução dos testes

```
int main(int argc, const char* argv[]) {
      clock_t start, end;
      double cpu_time_used;
      start = clock();
      //execute intended work load (e.g. a query)
      end = clock();
      cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
      printf("CPU Time:%f\n", cpu_time_used );
      return 0;
}
```

Snippet de código C para medição de tempo

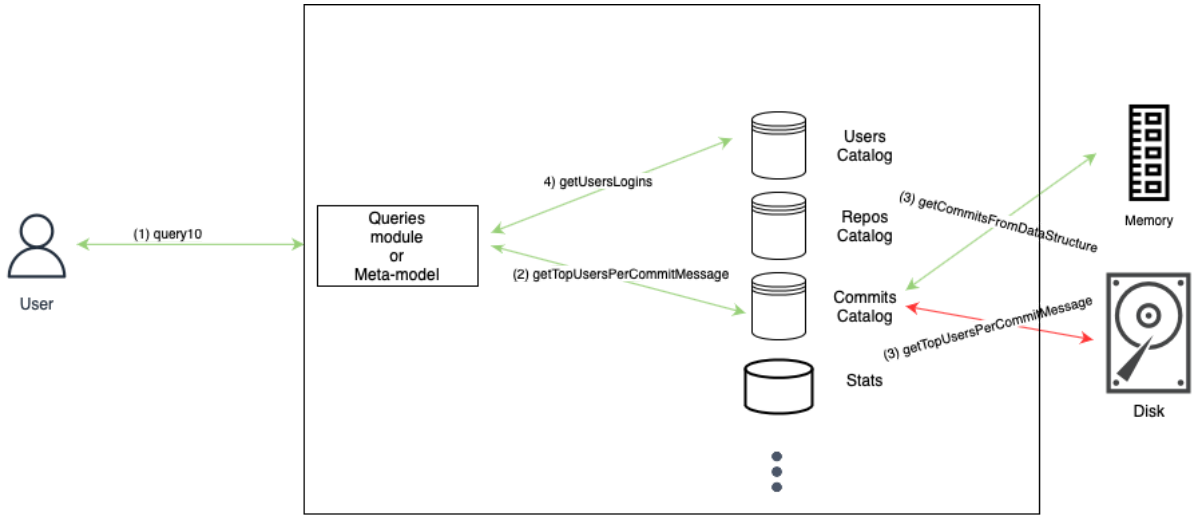
A Makefile do projeto deverá ser capaz de gerar um executável adicional de testes, capaz de invocar todos os testes e apresentar os seus resultados (a verificação de que cada uma das *queries* funciona conforme o especificado e respetivo tempo de execução). Este programa de testes deverá executar e validar a execução das funcionalidades do projeto, respeitando sempre a modularidade e encapsulamento dos componentes do sistema. O relatório do projeto deverá conter uma tabela com os resultados obtidos para as diferentes máquinas dos elementos de grupo, bem como as respectivas especificações do ambiente de execução. Naturalmente, os resultados deverão ser acompanhados de uma discussão que vise justificar os resultados obtidos.

Gestão de dados

Os ficheiros agora disponibilizados para este guião têm uma dimensão bastante superior aos utilizados nos guiões anteriores. De forma a manter padrões razoáveis de usabilidade e responder em tempo útil a funcionalidades pedidas pelo utilizador, poderá ser necessário trabalhar e organizar parte ou a totalidade dos dados em ficheiro. Desta forma, pretende-se que neste novo exercício este tipo de mecanismos seja implementado ao nível dos catálogos de dados desenvolvidos no guião 2 ou num ”meta-catálogo” que consiga gerir os dados em memória e/ou ficheiro consoante o volume de dados que está a tratar.
O principal objetivo deste exercício é então que se implemente um mecanismo eficiente de gestão de dados sem recurso a bases de dados e cujo funcionamento interno seja opaco para quem pretenda utilizar os módulos que implementam tal mecanismo. De forma a abstrair o código que invoque tais mecanismos do seu funcionamento, deverão ser aplicados os princípios do encapsulamento e modularidade abordados durante o semestre. Consequentemente, os catálogos e módulos que implementam as *queries* deverão manter o mesmo comportamento e funcionalidades disponibilizadas até agora (podendo ter um acréscimo de funcionalidades caso seja necessário), sendo o resto do programa alheio à nova mudança de comportamento.

O mecanismo de otimização pode fazer vários tipos de operações para melhorar a eficiência do programa. Este tipo de otimização pode ser efetuada a diferentes níveis: a nível da estruturação da informação em memória, a nível da execução das *queries* ou de leitura e carregamento dos dados no arranque do programa. Alguns exemplos de possíveis otimizações são:

- Armazenar resultados intermédios em ficheiro;
- Guardar resultados de *queries* em ficheiro;
- Guardar dados de forma estruturada e/ou ordenada segundo diferentes critérios em diferentes ficheiros;
- etc…



Exemplo **ilustrativo** de possíveis fluxo de execução da query 10

A figura acima ilustra 2 possíveis cenários de execução para a query 10: um cenário similar ao esperado no guião 2 e um novo cenário que visa ilustrar uma possível otimização desta funcionalidade. Segundo os requisitos do 2º guião, esta query deveria obter o top N de utilizadores com as maiores mensagens de *commit* por repositório. Esta informação deveria constar em memória, armazenada em estruturas de dados apropriadas para responder de forma eficiente às *queries* pedidas, tendo um fluxo de execução similar ao representado na figura a cor verde. Uma possível otimização a fazer neste novo guião poderia ser substituir o passo (3), em que o catálogo de commits obteria o top N de utilizadores com as maiores mensagens de *commit* por repositório de um ficheiro (processo ilustrado a cor vermelha). Este ficheiro poderia ser construído no momento do carregamento dos dados no arranque do programa ou após a primeira execução da query 10.

Os resultados da aplicação das otimizações deverão ser discutidos no relatório, confrontando estes resultados com os obtidos antes de efetuar a otimização.
A necessidade de implementação de um mecanismo mais complexo de gestão de dados depende da performance proporcionada pelo programa desenvolvido. Caso as estruturas de dados e algoritmos implementados para responder às *queries* tenham sido cuidadosamente concebidos, eventuais otimizações efetuadas neste sentido poderão até nem ser significativas. Nesse caso, a solução original deverá ser mantida, sendo os resultados também reportados no relatório.

Indicações para o relatório

Neste último guião, o relatório terá um maior peso na avaliação (20% da cotação do guião). Deste modo, é essencial não descurar esta componente de avaliação, algo que tem acontecido nos últimos guiões. De forma a clarificar o que se espera que conste no relatório, segue-se uma recomendação de estruturação e constituição para elaborar um relatório:

- Introdução:
    - descrever brevemente o problema, motivação e objetivos (aprox. 1 página);
- Solução técnica:
    - mudanças efetuadas em relação ao guião 2 de forma a incorporar os critérios de modularidade e encapsulamento (nos grupos em que se justifique). **Exemplos de aspetos a abordar:** criação de novos módulos, conceção dos *header files* e ficheiros de implementação de modo a implementar tipos opacos e a garantir encapsulamento de dados e especificação das APIs;
    - descrição da implementação para cada um dos exercícios, justificando as opções de implementação tomadas, soluções alternativas consideradas.
- Resultados e discussão:
    - Apresentação dos resultados dos exercícios, com especial ênfase nos testes efetuados;
    - Apresentar comparações de *performance* antes *versus* depois do exercício de otimização;
    - Justificação dos valores obtidos;
    - Apresentação de resultados em formato tabular, de preferência resultantes de várias medições efetuadas em diferentes máquinas. Exemplo de uma metodologia de medição de *performance*:

        - executar e medir 10x a funcionalidade/query pretendida;
        - descartar o valor máximo e mínimo obtidos;
        - calcular a média dos valores restantes.

      Nota: não é necessário reportar todos os tempos de execução; apenas o valor médio final calculado para cada *query*.
    - Especificação das condições do ambiente de execução (SO, CPU, memória, disco, etc) e parâmetros de *queries* utilizados;
- Conclusão:
    - Reflexões críticas acerca dos resultados produzidos;
    - Aspetos passíveis de serem melhorados.