Programação de Computadores Estruturada

(UFCD 5412)

2023 (TPSI 0922) – Projeto final de módulo

Entrega: A definir com os alunos

1. Enquadramento

Uma conhecida agência de viagens europeia (Gold Train), contratou a nossa empresa (KISS software) para a conceção e desenvolvimento de um portal que permita, em simultâneo, a venda de viagens e produtos da empresa, como também gerir o seu portfólio. Sendo uma empresa com uma dimensão assinalável possui um portfólio extenso, com destinos para mais de 500 cidades europeias.

Figura 1 - Ferrovia na Europa (tempos de ligação)

Conforme o próprio nome da empresa indica, dedica-se em exclusividade ao transporte ferroviário, tendo uma carteira de clientes variável, mas que, contudo, podem ser classificados e caracterizados em três (03) grupos distintos:

- Clientes empresariais, para quem o tempo é dinheiro e que usualmente procuram as soluções de deslocação mais rápida. Este grupo de cliente têm uma grande incidência na Europa Central, pois as deslocações por ferrovia acabam por ser mais rápidas que as aéreas, quando considerando os tempos de check-in e verificações de segurança nos aeroportos;
- Clientes regulares, que procuram usualmente as soluções mais económicas;
- Clientes turísticos, normalmente o setor sénior da população (+65) que procuram passeios turísticos, sendo a ferrovia uma ótima solução para apreciar a beleza natural da Europa;

2. O portal Gold Train

O portal desenhado pela nossa empresa assenta em num *front-end web,* destinado à interação com o cliente, e um conjunto de aplicações (algumas delas já existentes - legadas) que comunicam entre si através de uma Application Programming Interface (API). Apresenta-se na Figura 2, um diagrama genérico da arquitetura elaborada pela *KISS software*:

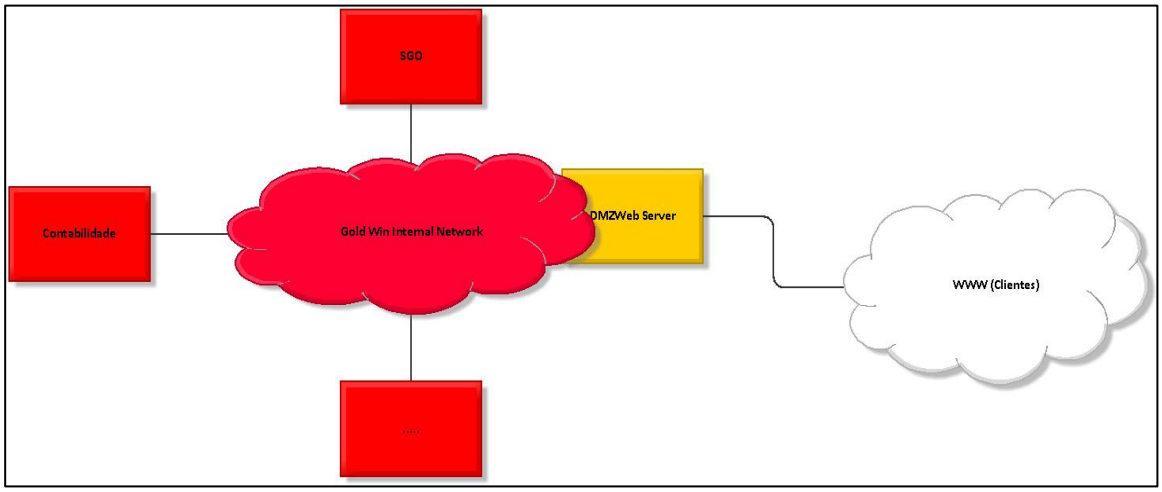

Figura 2 - Arquitetura GOLD Train

O *front-end*, designado por DMZ web server e que será disponibilizado aos clientes, irá comunicar com um conjunto de aplicações. O **Sistema de Gestão de Ofertas (SGO)** e o sistema de contabilidade também serão responsabilidade da nossa empresa. Para a realização deste projeto, a KISS software, criou várias equipas, sendo que à sua equipa foi designado o desenvolvimento do **SGO**.

A API, composta por uma série de funções, permite a comunicação entre as diferentes aplicações. Apesar da API ir funcionar através da rede IP interna da companhia,

nesta primeira faze pretende-se que a API seja implementada através da leitura e escrita do *Standard Input* (stdin) e *Standard Output* (stdout), respetivamente. A API SGO terá que ser escrita em C.

3. Especificação do programa

O SGO destina-se a guardar e processar informação referente às cidades, e ligações entre estas (por via férrea), que fazem parte do portefólio da GOLDEN train.

Cada cidade é composta, pelo menos, por um código de identificação (3 caracteres), um nome (MAX 50 carateres) e um estado (podem ser adicionadas outras características para facilitar a implementação da API). O estado de uma cidade poderá tomar um de dois valores: 1 (ativo) ou 0 (inativo) e destina-se a marcar cidades que por diversas razões devam ser excluídas temporariamente das rotas da companhia (e.g. situações de conflitos internos, ameaça terroristas, etc).

Cada ligação entre duas cidades (e.g. EEE -> FFF) é caracterizada por 3 atributos, todos representados por números reais maiores que zero:

- Índice Temporal: Representa as unidades de tempo da ligação;
- Índice Económico: Representa as unidades monetárias para a ligação;
- Índice Turístico: Representa uma avaliação da beleza natural (gosto pessoal dos clientes) do percurso. Esta avaliação é extraída do sistema de Inquéritos de Satisfação da GOLD Train, através de um algoritmo avançado que converte opiniões, valor de viagens e tempo das mesmas num índice. Quanto menor o valor do índice melhor a avaliação;

Um conjunto de ligações (uma ou mais) identifica uma **rota** entre duas cidades.

4. Sugestão de estrutura de dados

Um sistema deste tipo pode ser visto como um grafo orientado com pesos, onde cada vértice representa uma cidade e cada aresta representa uma ligação. Estes tipos de estruturas podem ser representados por matrizes de adjacência ou uma tabela de listas encadeadas. Sugere-se a leitura e consulta atenta do ficheiro grafos.pdf (slides passados nas aulas), onde poderá encontrar muita informação útil para a resolução deste problema.

5. Especificação da API

A API SGO irá implementar o seguinte conjunto de funções (entre parênteses a letra que identifica a função):

- Adiciona uma nova cidade (A);
- Altera o estado de uma cidade (O);
- Apaga uma cidade (P);
- Devolve Informação sobre a cidade (Y);
- Devolve o Número Total de Cidades (N);
- Adiciona uma ligação entre duas cidades (C);
- Apaga uma ligação entre duas cidades (I);
- Altera o Índice Turístico de uma ligação (T);
- Altera o Índice Económico de uma ligação (E);
- Altera o Índice Temporal de uma ligação (H);
- Melhor rota entre duas cidades (R);
- Guardar dados da aplicação (G)
- Lista todas as cidades (Z);
- Sair da aplicação (X)

A API implementa o seguinte formato de entrada:

`<Option> <param1> <param2> <…>`, onde:

- Option: Um carater que identifica uma função da API
- Param: parâmetro, sendo o seu número e tipo consoante cada função, os quais se apresentam de seguida;
- A opção e os parâmetros estão separados pelo carater espaço (‘ ‘);
- Assume-se que todos os comandos fornecidos à API SGO estão bem formatados, isto é, não serão fornecidos 3 argumentos quando a API designa para essa função dois argumentos;
- Nas especificações da API, abaixo, utiliza-se a seguinte convenção de cores (na aplicação não há cores) para melhor compreensão:
  - `Mensagens de INPUT;`
  - `Mensagens de OUTPUT;`
  - `Mensagens de ERRO (Output). Todos as mensagens de erro iniciam com o carater '*'.`

A: ADICIONA CIDADE

Input: `A <id> <nome>`

- `<id>`: String de 3 letras que identifica uma cidade;

- <nome>: -Nome da cidade que pode conter mais que uma palavra. String max 50 carateres;

Output: Em caso de sucesso não tem output. Em caso de a cidade já existir deverá ter como output o seguinte erro:

```
*Cidade <id> duplicada
```

Exemplo:

```
A LIS Lisboa
A POR Porto
A VFX Vila Franca de Xira
```

Outras indicações:

O: ALTERA O ESTADO DE UMA CIDADE

Input: `O <id> <estado>`

- <id>: 3 caracteres (letras) que identifica uma cidade;
- <estado>: 0 ou 1;

Output: Em caso de sucesso não tem output. Em caso de a cidade não existir deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Exemplo:

```
O LIS 1
O POR 0
O LIS 1
```

Outras indicações: Só devolve erro caso a cidade não exista. As cidades inativas não podem fazer parte das rotas, nem podem ser origem ou destinos.

P: APAGA UMA CIDADE

Input: `P <id>`

- <id>: 3 caracteres (letras) que identifica uma cidade;

Output: Em caso de sucesso não tem output. Em caso de a cidade não existir deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Exemplo:

```
P LIS
P POR
P LIS
*Cidade LIS inexistente
```

Outras indicações: Só dá erro caso a cidade não exista. As ligações de e para essa cidade são todas removidas.

Y: DEVOLVE INFORMAÇÃO SOBRE A CIDADE

Input: `Y <id> <type>`

- <id>: 3 caracteres (letras) que identifica uma cidade;
- <type>: 1 para informação completa, 0 para informação reduzida

Output: Em caso de sucesso produz o seguinte output:

Type=1

```
  <id> (<estado>):  Existem <num> ligações a partir
de <nome>
  \t-><id_d>: <i_H> <i_E> <i_T>
```

Type=0

`<id> (<estado>): Existem <num> ligações a partir de <nome>`

- `<id>`: 3 caracteres (letras) que identifica uma cidade;
- `<estado>`: Estado da cidade (0 ou 1);
- `<id_d>`: 3 caracteres da cidade destino
- `<i_H>`: Índice Temporal (999.99 > real >0);
- `<i_E>`: Índice Económico (999.99 > real > 0);
- `<i_T>`: Índice Turístico (999.99 > real > 0);

No caso de a cidade não existir deverá ter como output o seguinte erro:

`*Cidade <id> inexistente`

Exemplos:

```
Y LIS 0
LIS (1): Existem 3 ligações a partir de Lisboa
Y LIS 1
LIS (1): Existem 3 ligações a partir de Lisboa
     ->POR: 5.00 3.55 6.45
     ->PAR: 6.45 8.33 2.34
     ->UHF: 6.45 9.33 2.10
Y XXX 1
*Cidade XXX inexistente
```

Outras indicações: A lista das ligações para cidades (type=1) devem ser apresentadas por ordem de entrada (escrever primeiro as que foram inseridas primeiro). Se uma ligação for apagada e novamente inserida, passará para último. Os valores dos índices devem ser apresentados com duas casas decimais.

N: DEVOLVE O NÚMERO TOTAL DE CIDADES

Input: `N`

Output: devolve o número inteiro que representa todas as cidades

Exemplo:

```
N
5
```

Outras indicações: As cidades inativas (estado=0) também são consideradas para a contagem.

C: ADICIONA UMA LIGAÇÃO ENTRE DUAS CIDADES

Input: `C <id_origem> <id_destino>`

- `<id_origem>`: 3 caracteres (letras) que identifica uma cidade de origem;
- `<id_destino>`: 3 caracteres (letras) que identifica uma cidade de destino;

Output: Em caso de sucesso não tem output. Em caso de a cidade (ou ambas) não existir deverá ter como output o seguinte erro:

`*Cidade <id> inexistente`

Exemplo:

```
C LIS POR
C POR LIS
```

```
C LIS PAR
*Cidade PAR inexistente
C PAR XPT
*Cidade PAR inexistente
*Cidade XPT inexistente
```

Outras indicações: Só devolve erro caso a cidade não exista (ou ambas). Se ambas não existirem deve ser apresentado uma mensagem de erro por cada uma. Na criação de uma ligação todos os índices devem ser inicializados a um (1). Para simplificação, pode assumir que não há mais que uma ligação no mesmo sentido entre duas cidades (e.g. não são criadas duas ligações de lisboa para o Porto do tipo 2 x LIS->POR, mas pode haver uma LIS->POR e outra POR->LIS);

I: APAGA UMA LIGAÇÃO ENTRE DUAS CIDADES

Input: `I <id_origem> <id_destino>`

- `<id_origem>`: 3 caracteres (letras) que identifica a cidade de origem;
- `<id_destino>`: 3 caracteres (letras) que identifica a cidade de destino;

Output: Em caso de sucesso não tem output. Em caso de uma cidade não existir (ou ambas) deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Caso ambas as cidades existam, mas a ligação não, deverá dar a seguinte mensagem de erro:

```
*Ligação <id_origem>-><id_destino> inexistente
```

Exemplo:

```
I LIS POR
I POR SXC
*Cidade SCX inexistente
I LIS FAR
*Ligação LIS->FAR inexistente
```

Outras indicações: Caso ambas as cidades não existam deverá apresentar ambos os erros (primeiro referente à origem e depois ao destino). Caso uma ou ambas as cidades não existam, não apresenta o erro de não haver a ligação.

T: ALTERA O ÍNDICE TURÍSTICO DE UMA LIGAÇÃO

Input: `T <id_origem> <id_destino> <it>`

- `<id_origem>`: 3 caracteres (letras) que identifica a cidade de origem;
- `<id_destino>`: 3 caracteres (letras) que identifica a cidade de destino;
- `<it>`: Novo índice turístico (real >0)

Output: Em caso de sucesso não tem output. Em caso de uma cidade não existir (ou ambas) deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Caso as cidades existam, mas a ligação não, deverá dar a seguinte mensagem de erro:

```
*Ligação <id_origem>-><id_destino> inexistente
```

Exemplo:

```
T LIS POR 4.56
T POR SXC 3.33
*Cidade SCX inexistente
T LIS FAR 9.00
*Ligação LIS->FAR inexistente
```

Outras indicações: Caso uma ou ambas as cidades não existam, apresenta erro de cidade inexistente. Caso ambas as cidades não existam deverá apresentar ambos os erros (primeiro referente à origem e depois ao destino). Caso uma ou ambas as cidades não existam, não apresenta o erro de não haver a ligação.

E: ALTERA O ÍNDICE ECONÓMICO DE UMA LIGAÇÃO

Input: `E <id_origem> <id_destino> <ie>`

- `<id_origem>`: 3 caracteres (letras) que identifica a cidade de origem;
- `<id_destino>`: 3 caracteres (letras) que identifica a cidade de destino;
- `<ie>`: Novo índice económico (real >0)

Output: Em caso de sucesso não tem output. Em caso de uma cidade não existir (ou ambas) deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Caso as cidades existam, mas a ligação não, deverá dar a seguinte mensagem de erro:

```
*Ligação <id_origem>-><id_destino> inexistente
```

Exemplo:

```
E LIS POR 4.56
E POR SXC 3.33
*Cidade SCX inexistente
E LIS FAR 9.00
*Ligação LIS->FAR inexistente
```

Outras indicações: Só dá erro caso a cidade não exista. Caso ambas as cidades não existam deverá apresentar ambos os erros (primeiro referente à origem e depois ao destino). Caso uma ou ambas as cidades não existam, não apresenta o erro de não haver a ligação.

H: ALTERA O ÍNDICE TEMPORAL DE UMA LIGAÇÃO

Input: `H <id_origem> <id_destino> <ih>`

- `<id_origem>`: 3 caracteres (letras) que identifica a cidade de origem;
- `<id_destino>`: 3 caracteres (letras) que identifica a cidade de destino;
- `<ih>`: Novo índice temporal (real >0)

Output: Em caso de sucesso não tem output. Em caso de uma cidade não existir (ou ambas) deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Caso as cidades existam, mas a ligação não, deverá dar a seguinte mensagem de erro:

```
*Ligação <id_origem>-><id_destino> inexistente
```

Exemplo:

```
H LIS POR 4.56
H POR SXC 3.33
*Cidade SCX inexistente
H LIS FAR 9.00
*Ligação LIS->FAR inexistente
```

Outras indicações: Só dá erro caso a cidade não exista. Caso ambas as cidades não existam deverá apresentar ambos os erros (primeiro referente à origem e depois ao destino). Caso uma ou ambas as cidades não existam, não apresenta o erro de não haver a ligação.

R: MELHOR ROTA ENTRE DUAS CIDADES

Input: `r <id_origem> <id_destino> <indice>`

- `<id_origem>`: 3 caracteres (letras) que identifica a cidade de origem;
- `<id_destino>`: 3 caracteres (letras) que identifica a cidade de destino;
- `<indice>`: Um caracter ('H','E' ou 'T') que indica o índice a considerar no cálculo:
  - H: A melhor rota é aquela com **o menor** índice temporal. O índice temporal representa o tempo gasto no trajeto;
  - E: A melhor rota é aquela com **o menor** índice económico. O índice económico representa as unidades monetárias para um trajeto;
  - T: A melhor rota é aquela com **o menor** índice turístico, contudo;

Output: Em caso de sucesso apresenta o seguinte Output.

```
<id_origem>-><id_destino> <indice>=<valor_ indice>
  \t-><id_origem>-><id_1>-><id_N>-><id_destino>
```

- `<id_1>`: 3 caracteres (letras) que identifica a 1ª cidade no percurso, após a cidade de origem;
- `<id_N>`: 3 caracteres da cidade Nª do percurso.
- `<valor_indice>`: Soma dos índices de todas as ligações do caminho;

Em caso de uma cidade não existir (ou ambas) deverá ter como output o seguinte erro:

```
*Cidade <id> inexistente
```

Em caso de uma cidade estar inativa (cidade origem ou destino) deverá ter como output o seguinte erro:

```
*Cidade <id> inativa
```

Caso não exista ligação entre a cidade de origem e destino deverá apresentar o seguinte erro:

```
*<id_origem>-><id_destino> sem ligação
```

Exemplo:

```
R LIS POR H
LIS->PAR H=15.78
     LIS->POR
```

```
R LIS PAR E
LIS->PAR E=45.78
     LIS->BAR->BRD->PAR
R LIS FAR H
*Cidade FAR inexistente
R LIS FIR T
*Cidade FIR inativa
R PAR LND T
*PAR->LND sem ligação
```

Outras indicações: Caso ambas as cidades não existam deverá apresentar ambos os erros (primeiro referente à origem e depois ao destino). Caso exista mais que uma rota com o mesmo valor, devem ser todas escritas, por ordem alfabética da rota. Tal como o caso abaixo:

```
R LIS PAR H
LIS->PAR E=55.78
     LIS->BAR->BRD->PAR
     LIS->MAD->BRD->PAR
```

Z: LISTA TODAS AS CIDADES

Input: `Z`

Output: Apresenta, para cada cidade, a sua informação resumida (comando `Y <id> 0)`, sendo na listagem as cidades apresentadas por ordem alfabética:

Caso não exista nenhuma cidade para apresentar erro:

```
*Base de dados vazia
```

Exemplo:

```
Z
LIS (1): Existem 3 ligações a partir de Lisboa
POR (1): Existem 2 ligações a partir de porto
ROM (0): Existem 4 ligações a partir de Roma
ZUR (1): Existem 2 ligações a partir de Zurique
```

Outras indicações: Mesmo as cidades inativas devem ser impressas e contabilizadas as suas ligações (embora essas ligações não possam ser usadas no cálculo de rotas).

G: GUARDA OS DADOS DA APLICAÇÃO

Input: `G`

Output: Lista de comandos A, C, T, E, H gravados de forma sequencial (um por linha) num ficheiro de texto, cujo caminho é passado através de um argumento de linha de comandos. Caso esse argumento não tenha sido passado, gravar no ficheiro data.sgo (cria caso não exista) dentro da diretoria da aplicação. Após gravação apresenta a seguinte output:

```
SGO gravado no ficheiro <nome_fich>
```

Caso seja passado por linha de comando um ficheiro que não tenha a extensão .sgo a aplicação utiliza para gravar data.sgo e apresenta o seguinte erro ao iniciar:

```
*Ficheiro <nome_ficheiro> não suportado
```

Exemplo (Executando a aplicação com "./SGO sgo.txt"):

```
*Ficheiro sgo.txt não suportado
G
SGO gravado no ficheiro data.sgo
```

Outras indicações: Não é avaliado nem testado a forma como se grava os dados no ficheiro. O formato e a ordem em que os comandos são escritos no ficheiro são uma opção sua.
Contudo, sugere-se que grave os dados usando a sintaxe definida neste documento para a API e relembra-se que para restabelecer o estado original da informação a ordem dos comandos interessa.
A aplicação ao iniciar e caso seja passado um argumento (caminho para o ficheiro) tem que restabelecer a sua base de dados (executar os comandos no ficheiro), caso os dados existam no ficheiro. Será nesse mesmo ficheiro gravada a informação através do comando G, apagando a que existia previamente (caso exista) e rescrevendo. Caso o nome do ficheiro seja passado mas este não exista, deverá ser criado para se gravar os dados.
A aplicação não permite outros ficheiros que não correspondam ao seguinte formato *.sgo;

X: SAIR DA APLICAÇÃO

Input: `X`
Output: Sem output
Exemplo:

```
X
```

Outras indicações: A aplicação ao terminar tem que libertar a memória em utilização;

6. Execução e Testes do Programa

O programa deve ser compilado através do makefile disponível na pasta do projeto, sendo que o executável tem que ter o nome SGO_API (já está previsto no makefile). Só tem necessidade de editar o ficheiro makefile para inserir referência a ficheiros objeto (.o) e bibliotecas (.h) que para organização do seu projeto necessite de criar. Para compilar o seu projeto deverá utilizar o seguinte comando:

```
make
```

Para testar o seu programa possui no projeto uma pasta com vários testes. Pode testar todos os testes executando o script também na pasta do projeto (test_all.sh). Ou pode executar os testes um a um:

```
$ ./SGO_API < ./testes/test01_1.in > test01_1.myout
```

Posteriormente poderá comparar o seu output com o output previsto usando o comando **`diff`**

```
$ diff ./testes/test01_1.out test01_1.myout
```

Após a estruturação do modelo de dados e programa (**parte mais importante**), sugere-se vivamente que construa e teste o seu projeto pela ordem de numeração dos testes, conforme a tabela abaixo. Esta ordem reflete o modo como vão ser avaliados os testes, deixando-se os comandos mais complexos para o fim.

Testes	Funcionalidades
`test00_*.in`	X (Sair da aplicação)
`test01_*.in`	A, Z, X, N (Inserir cidade, listar cidade, número cidades)
`test02_*.in`	A, Z, X, N, O (+ Altera estado cidade)
`test03_*.in`	A, Z, X, N, O, C, Y (+ Cria Ligação, Info detalhada cidade)
`test04_*.in`	A, Z, C, Y, X, I (+ Apaga Ligação entre cidades)
`test05_*.in`	A, Z, C, Y, X, T, E, H (+ Altera índices das ligações)
`test06_*.in`	A, Z, C, Y, X, I, P (+ Apaga cidade)
`test07_*.in`	A, Z, C, Y, X, I, P (+ Guarda info em ficheiro) (**Nota**: test07_01.in manda guardar e teste test07_02.in deve ser executado passando esse ficheiro para que se possa fazer o load e executar comandos de listagem)
`test08_*.in`	Todas as funcionalidades mais R (Melhor rota)

7. Entrega do Projeto

A entrega do projeto deverá respeitar o procedimento seguinte:

- No servidor Git está criado um projeto para cada grupo como um conjunto de ficheiros iniciais;
- A entrega é feita unicamente através do servido, sendo que o acesso ao projeto final vai ser cortado exatamente às **23H59 do dia 17 de março de 2020**;
- O projeto vai ter que compilar sem erros através do comando "make", **não sendo avaliados projetos em que tal não aconteça**.
- Até à data limite poderá efetuar o número de submissões que desejar (git push), sendo utilizada para efeitos de avaliação a última submissão efetuada. Deverá, portanto, verificar cuidadosamente que a última submissão corresponde à versão do projeto que pretende que seja avaliada. **Não existirão exceções a esta regra**.

8. Avaliação do Módulo e do Projeto

a. Avaliação do Projeto

(a) 12 valores -> testes automáticos;

(b) 6 valores -> Qualidade código e outros aspetos (comentários, organização, opções);

(c) 2 valores -> Gestão da memória (valgrind)

(d) Até -6 valores -> Entrevista individual Opcional;

b. Componentes da Avaliação

Na avaliação do projeto serão consideradas as seguintes componentes:

- A primeira componente avalia o desempenho da funcionalidade do programa realizado, através de testes automáticos. Esta componente é avaliada entre 0 e 12 valores. Os testes serão fornecidos após a avaliação.
- A segunda componente avalia a qualidade do código entregue, nomeadamente os seguintes aspetos: comentários, indentação, estruturação, modularidade, abstração, entre outros. Esta componente vale 6 valores relativamente à classificação calculada no item anterior.
- Durante a discussão final do projeto (opcional e se o professor entender que terá que haver ou tiver dúvidas sobre a participação de elementos dos grupos) será averiguada a participação de cada elemento do grupo na realização do projeto, bem como a sua compreensão do trabalho realizado. A respetiva classificação será ponderada em conformidade, isto é, elementos do mesmo grupo podem ter classificações diferentes. Ou seja, será dada uma penalização de 0 valores até 6 valores consoante a entrevista. **Elementos do grupo em que se verifique claramente não terem participado na realização do respetivo projeto, não tendo a mínima ideia como o projeto está estruturado, terão a classificação final de 0 (zero) valores no projeto**.

9. Considerações finais

a. Não deixe a execução do projeto para os últimos dias. *Normalmente o que pode correr mal, corre, da pior maneira possível e na pior altura possível* (Lei de Murphy).
b. Faça as coisas com tempo e coloque as dúvidas. Todas as dúvidas são pertinentes e devem ser colocadas no grupo criado para o efeito. As dúvidas de um podem ser de outros.
c. Qualquer lapso, ambiguidade ou erro encontrado no enunciado ou testes, que considere que tenha ocorrido, deve ser comunicado de imediato através do grupo para que o formador reveja ou revele o seu entendimento sobre essa questão. **Sempre que tenha dúvidas pergunte**….
d. Apesar de serem fornecidos alguns testes, eles são uma pequena percentagem dos que irão ser efetuados e não representam todos os cenários possíveis. **É da sua responsabilidade testar exaustivamente as funcionalidades da aplicação** e criar mais testes para o efeito. Ou seja, a passagem em todos os testes fornecidos não garante a nota máxima no projeto (mas é muito bom sinal).

Bom trabalho!