Trabalho de aprofundamento AP2

André Patacas, Gil Teixeira

Aplicação para o cálculo de Largura de Banda e de latência

DETI

André Patacas, Gil Teixeira
(93357) andrepatacas@ua.pt, (88194) gilteixeira@ua.pt

9 de Abril de 2019

Conteúdo

Resumo

Este relatório serve para descrever uma ferramenta desenvolvida para calcular a largura de banda e a latência da máquina onde a aplicação se encontra a correr. Calculam-se estest valores para um determinado servidor ou para um conjunto, de cardinalidade especificável, de servidores de um país, também especificável. No final a aplicação cria um relatõrio, em csv, e assina-o com a chave privada fornecida por um ficheiro à parte pelo utilizador.

Capítulo 1

Introdução

A aplicação foi desenvolvida em python3 no âmbito da disciplina de Laboratórios de Informática, no ano letivo 2018/2019. A adicionar às especificações básicas pedidas, segundo o guião sobre regras do segundo trabalho de aprofundamento, construi-se ainda suporte para pydocs para haver uma explicação mais detalhada sobre cada método e classe no nosso projeto. O programa foi escrito com base em test driven development (Capítulo 2) e, como tal, os testes unitários e funcionais foram criados primeiro, seguidos por um esqueleto do programa e finalmente por vários updates a ambos (chap.resultados) para chegar ao estado em que a aplicação se encontra de momento (**??**). Finalmente são tiradas as conclusões sobre os aspetos positivos e, potencialmente, negativos desta solução em concreto (**??**)

Capítulo 2

Metodologia

1. Criar o esqueleto do programa que é agora o inicializador da classe (labi02) se esta for chamada diretamente;
2. Criar o ficheiro test_labi_02 como um teste que, apenas se a construção da aplicação for robusta e exatamente como especificada, passa.
3. Criar o programa labi_02 e definir as funções com os argumentos de entrada e cada uma com uma descrição detalhada, disponivel nos pydocs, dos aspetos funcionais de cada função.
4. Ajustar os métodos de forma a que a aplicação passa todos os testes impostos no teste criado.
5. Testar o programa manualmente e/ou com testes funcionais.
6. Corrigir eventuais erros.
7. Iterar o processo de debugging e correção de erros.

Capítulo 3

Aplicação de Speed Test

3.1 Labi_02

Esta é a aplicação que foi desenvolvida e que pode ser utilizada diretamente de acordo com o usage demonstrado ao correr a aplicação sem argumentos. Toda a descrição feita neste relatório remete na mesma para a documentação, esta criada a quando do desenvolvimento da aplicação.

3.1.1 main

Este método é utilizado quando a app é chamada como classe principal (*main*). Inicialmente chama o método validate (Subseção 3.1.10), seguido de run_tests (Subseção 3.1.9), depois o report Subseção 3.1.8 com o resultado da anterior e finalmente assina esse relatório com o método create_signed_document (Subseção 3.1.5).

3.1.2 calc_download()

Cálculo da largura de banda.
Este método pede, inicialmente, para fazer um download de 100 megabytes ao target server dentro de 10 segundos. Depois verifica que não há mais data para ser recebida do *target_server* e finalmente calcula o time download 1mb que a máquina demora a fazer download de 1 megabyte.
Argumentos: *target_server*(dicionário com informação sobre o target server).
Retorna: (float) 1/*time_download_1mb*

3.1.3 calc_latency()

Cálculo da latência.
Este método tenta trocar dez comandos PING-PONG com o targe-server e calcula o tempo médio em milisegundos entre estas trocas.

Argumentos: target server (dicionário com informação sobre o target server).
Retorna:(int) *average_trade_time* em ms.

3.1.4 country_test()

Este método serve para calcular a largura de banda e latency da conexção a um servidor random do país passado como argumento.
Argumentos: (str) *target_country*.
Retorna: objeto *SpeedTestResult* com as informações relativas aos resultados do teste.

3.1.5 create_signed_document

Este método gera um *signature file* assinando o *report* com a chave privada no *key_path* especificado.
Argumentos:

1. *key_path* (str): The path to the file that contains the key;
2. *report_name* (str): Nome do *report* a ser assinado;
3. *signature_name* (str): Nome do *signature file* que será gerado.

Retorna: *None*.

3.1.6 id_test()

Este método serve para calcular a largura de banda e latency da conexão a um servidor com o id passado como argumento.
Argumentos: (int) *target_id*.
Retorna: objeto *SpeedTestResult* com as informações relativas aos resultados do teste.

3.1.7 random_test()

Este método serve para calcular a largura de banda e latency da conexção a um servidor random.
Argumentos: *None*.
Retorna: objeto *SpeedTestResult* com as informações relativas aos resultados do teste.

3.1.8 report()

Este método vai gerar um *test_report* basiado numa lista de objetos *SpeedTestResult* passados como argumentos.
Argumentos:

1. List[objeto *SpeedTestResult*];

2. *report_name* (str) - nome do ficheiro a ser gerado.

Retorna: *None*. O ficheiro *test_report* será gerado

3.1.9 run_tests()

Este método serve para calcular a largura de banda e latency da conexção a um *num* de servidores num país ou a um server com o id passado por argumento. Nota: se o terceiro argumento for um id a função realizará *num* testes a esses servidor, se for um país fará *num* testes usando a função Subseção 3.1.4 e se não foi passado terceiro argumento realiza um teste random (Subseção 3.1.7).
Argumentos:

1. *inteval*: intervalo de tempo entre cada teste realizado;
2. *num*: número de testes a realizar;
3. *id_or_country*: país (str) ou id (int) de um server;

Retorna: objeto *SpeedTestResult* com as informações relativas aos resultados do teste.

3.1.10 validate()

Este método trata da validação dos argumentos passados pela variávle sys.argv.
Argumentos: *None*.
Retorna: *None*.

3.1.11 usage()

Este método imprime a mensagem de erro passada como argumento e imprime a ajuda para utilização da aplicação.
Argumentos: *message* (str).
Retorna: *None*.

3.1.12 log()

Este método imprime a mensagem de passada como argumento com a cor passada como argumento.
Argumentos:

1. *message* (str);
2. *colour* (str).

Retorna: *None*.

3.1.13 log_error()

Este método chama Subseção 3.1.12 com a mensagem igual à passada como argumento mas com cor vermelho.
Argumentos: *message* (str).
Retorna: *None*.

3.1.14 log_warning()

Este método chama Subseção 3.1.12 com a mensagem igual à passada como argumento mas com cor amarela.
Argumentos: *message* (str).
Retorna: *None*.

3.1.15 log_verbose()

Este método chama Subseção 3.1.12 com a mensagem igual à passada como argumento mas com cor verde se o modo *verbose* estiver ativado.
Argumentos: *message* (str).
Retorna: *None*.

3.1.16 load_server()

Este método lê o ficheiro *"servers.json"* e cria um dicionário global com a lista de servidores. **Argumentos**: *None*.
Retorna: *None*.

Capítulo 4

Resultados

Capítulo 5

Análise

Capítulo 6

Conclusão