

# Gestão de Redes

28 de Fevereiro de 2021

## **Trabalho Prático 2**

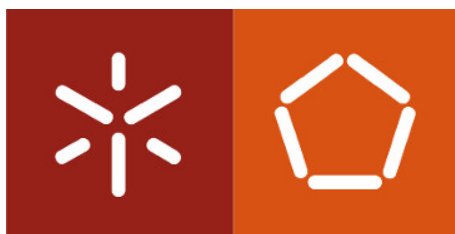
a83899

André Moraes

---

*Ferramenta de Monitorização*

---



Mestrado Integrado em Engenharia Informática  
Universidade do Minho

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Concepção/Desenho da resolução</b>	<b>3</b>
<b>3</b>	<b>Descrição da arquitetura da ferramenta</b>	<b>4</b>
3.1	SNMP Monitor . . . . .	4
3.2	SNMP Client . . . . .	5
3.3	Monitor . . . . .	6
3.4	Notifier . . . . .	6
3.4.1	Ficheiro de configuração . . . . .	6
3.5	Interface . . . . .	7
<b>4</b>	<b>Conclusão</b>	<b>9</b>

# 1 Introdução

Neste trabalho prático, o objetivo era implementar um programa para monitorização e análise de utilização dos recursos do sistema local pelos processos ativos num qualquer host. Os resultados são mostrados numa interface web, por forma a facilitar a sua visualização a utilizadores como gestores de redes.

Nestas próximas secções, aquilo que pretendemos é explicar cada um dos passos para a conceção deste projeto e para a sua estruturação.

## 2 Concepção/Desenho da resolução

Nesta secção será feita uma abordagem à arquitectura da aplicação, bem como às suas componentes e à forma como elas se relacionam.

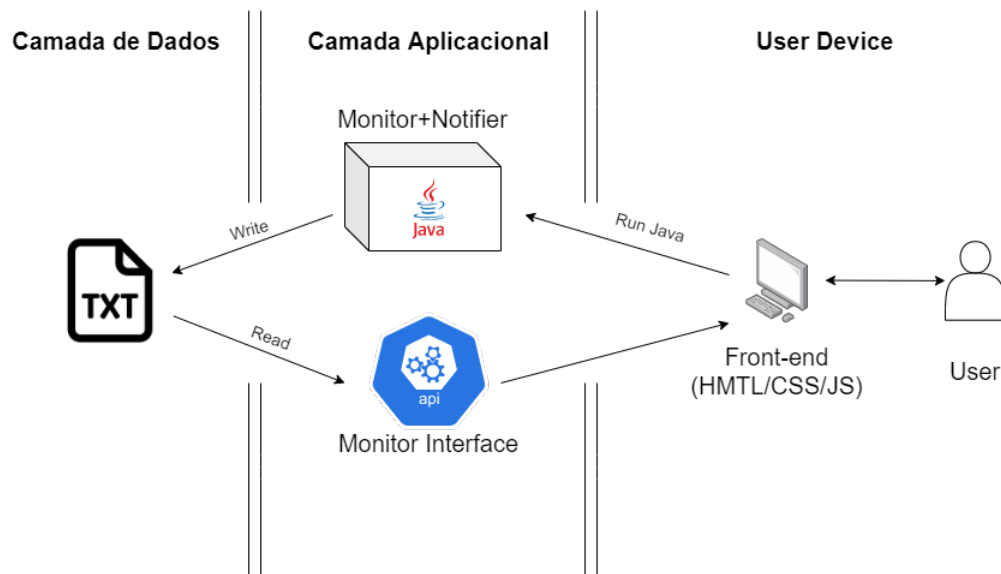


Figura 1: Modelo de Sistema

## 3 Descrição da arquitetura da ferramenta

### 3.1 SNMP Monitor

O programa foi desenvolvido em **Java**. Previamente, foi necessária a instalação de um agente **SNMPv2c** e uso de libraries externas como o **SNMP4J**, **JAVAMAIL** e **ConfigParser**.

O armazenamento dos dados é feita em ficheiros .txt. É necessário ter as pastas denominadas **cpu** e **ram** dentro da diretoria **logs** para o funcionamento deste. Também, dentro desta diretoria tem um ficheiro config.

Este programa está composto em 3 classes:

- **SNMP Client**
- **Monitor**
- **Notifier**

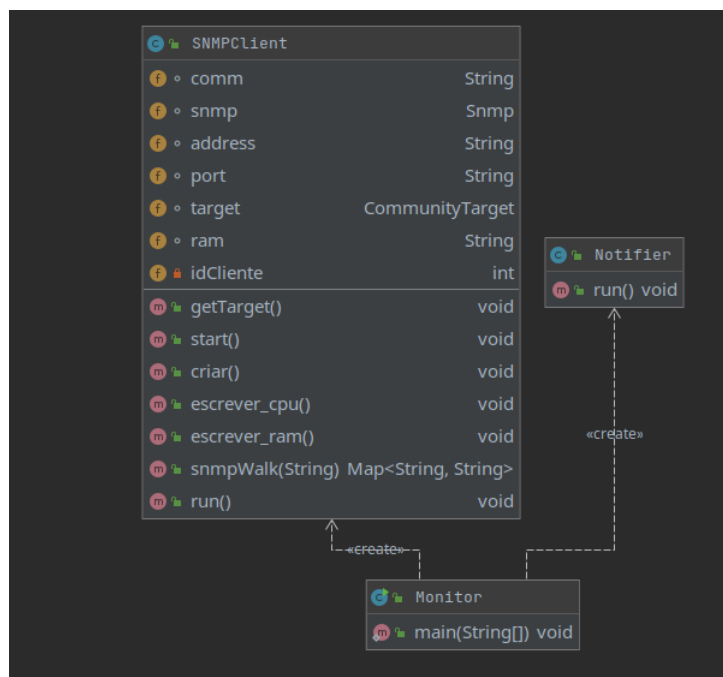


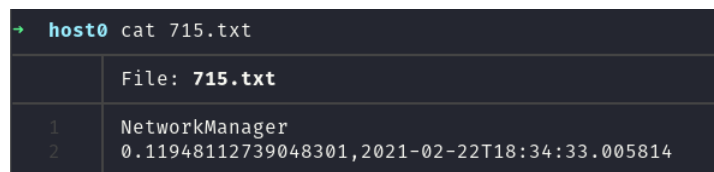
Figura 2: Diagrama de Classes

## 3.2 SNMP Client

Esta classe foi criada de modo a guardar os dados necessários por forma a facilitar a implementação da interface.

De seguida, guardo todos os processos numa *Hash-Map* onde a key é o OID que estava relacionado com o nome do processo. Depois de ter obtido os processos, são criadas duas diretorias para cada host, nas pastas ram e cpu e em cada uma destas diretorias são também criados um ficheiro para cada processo, apelidadas com o seu PID. Dentro de cada ficheiro é escrito o nome do processo.

Por último, foram criados dois métodos em que, um percorre a nossa *Hash-Map* com a quantidade de cpu através dos oid's, povoando os ficheiros .txt respetivos, e outro que faz exatamente a mesma coisa mas em vez de ser cpu, é de ram, escrevendo nos ficheiros a percentagem de ram gasta e o tempo local, como mostra a imagem seguinte:



```
→ host0 cat 715.txt
```

	File: 715.txt
1	NetworkManager
2	0.11948112739048301,2021-02-22T18:34:33.005814

Figura 3: Diagrama de Classes

Estes métodos estão constantemente a correr, com um pooling de 30 segundos.

Foi necessário recorrer a três tabelas distintas para a recolha de informação:

- **PID & Nome** : .1.3.6.1.2.1.25.4.2.1.2
- **CPU** : .1.3.6.1.2.1.25.5.1.1.1
- **RAM** : .1.3.6.1.2.1.25.5.1.1.2

### 3.3 Monitor

Nesta classe faço o parser dos dados da config e cria duas threads consoante o número de host's, uma para monitorizar os hosts e outra para notificar.

### 3.4 Notifier

Esta classe faz primeiramente parser do email do utilizador, do PID do processo que quer monitorizar e do limite máximo percentual de utilização de ram desse mesmo processo. Foi criada uma função que encontra o ficheiro e verifica se o último valor escrito nesse ficheiro é maior ou não do valor estabelecido. Se este for maior, é enviado um email para o utilizador. De referir que apenas funciona para **gmail.com**.

#### 3.4.1 Ficheiro de configuração

Para o bom funcionamento deste programa é necessário a criação de uma secção para cada user.

No ficheiro de configuração cada host tem de ter uma secção denominada host concatenado com o seu índice como exemplo: 'host0' e dentro da sua secção ter os campos descrição, address, port, community, ram, email, proc, max dos seus valores. Os valores 'ram' são referentes ao total de ram do seu computador, 'proc' é o PID do processo que vai ser monitorizado e o 'max' é o valor limite percentual de ram do processo 'proc'

Exemplo:

```
[host0]
address = 127.0.0.1
port = 161
community = gr2020
ram = 16270352
email = 123teste@gmail.com
proc = 1756
max = 10
```

### 3.5 Interface

Para a realização da interface foi utilizada a ferramenta **Express.js**, devido à familiaridade com esta ferramenta. Do mesmo modo foi usado também o **fs** para efetuar leituras dos ficheiros e o **pug** para as páginas html.

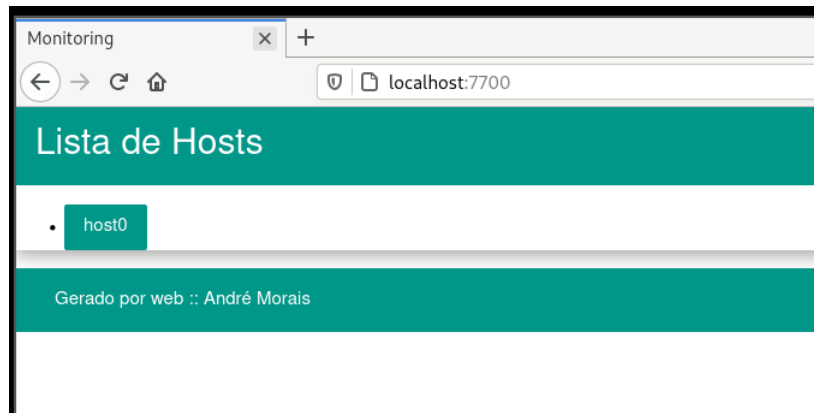


Figura 4: Página Inicial

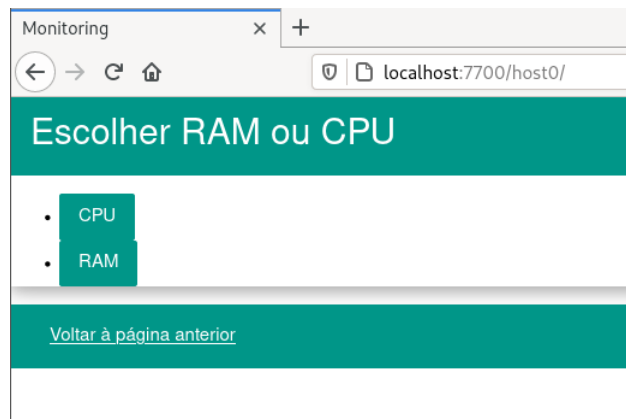


Figura 5: Página do Host



Monitoring x +

localhost:7700/ram

### Lista de Processos (RAM)

PID	Nome	Percentagem Ocupada	
1443	java	8.555438751417302	Gráfico
986	firefox	4.908166707149298	Gráfico
1236	WebExtensions	1.821447993257921	Gráfico
2039	Discord	1.7372457584199776	Gráfico
1294	Web Content	1.2659590892686279	Gráfico
1130	Privileged Cont	0.9605446765994983	Gráfico
1961	Discord	0.8798088695315257	Gráfico
1990	Discord	0.6485415927080127	Gráfico
943	Xorg	0.6158440825373661	Gráfico
4571	Web Content	0.5762383014208912	Gráfico
2008	Discord	0.4482509044672174	Gráfico

Figura 6: Página da lista de processos - RAM

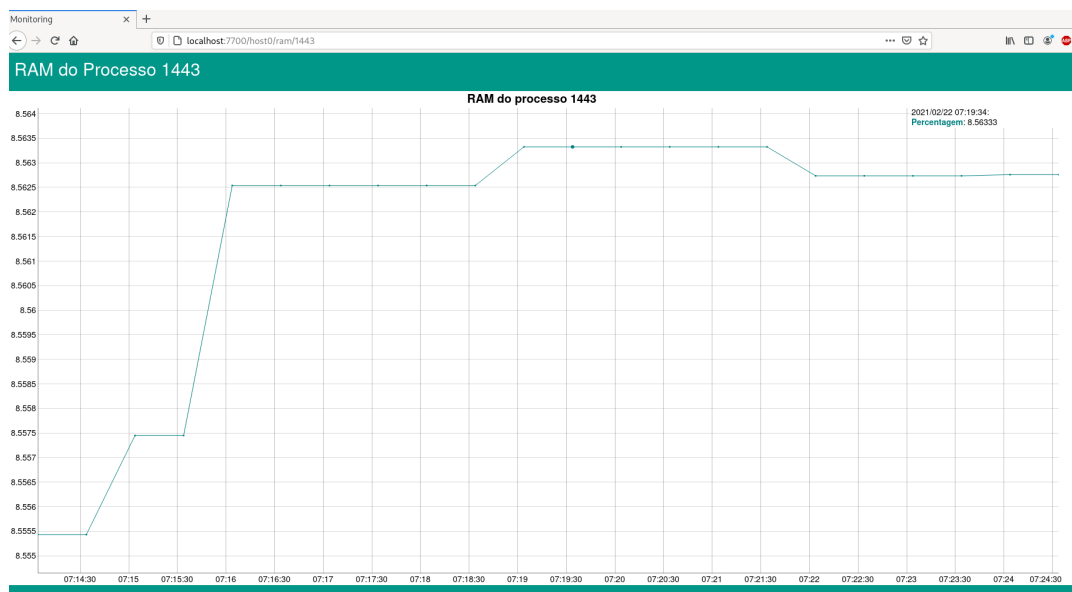


Figura 7: Página com gráfico de um processo

## 4 Conclusão

Em suma a realização deste projeto permitiu-nos consolidar a aprendizagem da UC de Gestão de redes. Todos os objetivos foram atingidos e os extras implementados.

Havia espaço para melhorias como por exemplo o nome dos ficheiros terem o nome do processo e não do pid, para ser mais fácil a sua chamada na config, ou também escrever a quantidade de cpu gasta em percentagem.

## Referências

- [1] Exemplo de um SNMPWALK em java:  
<https://examples.javacodegeeks.com/enterprise-java/snmp4j/snmp-walk-example-using-snmp4j/>