UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE ENGENHARIA DE COMPUTAÇÃO

GABRIEL TEODORO COBLINSKI HRYSAY

RELATÓRIO FINAL
DASHBOARD PARA LINUX

CURITIBA
2022

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Os sistemas operacionais baseados em Linux é frequentemente escolhido por profissionais da computação e de áreas relacionadas devido à sua grande versatilidade e modalidade de desenvolvimento e implementação *Open Source*. Para esses profissionais, também é de grande importância conhecer e entender o funcionamento destes sistemas, bem como a utilização dos recursos de uma máquina por esses sistemas. Com o objetivo de apresentar algumas estatísticas e informações sobre o hardware e a utilização dos principais recursos, desenvolvel-se um *software* que os apresenta em uma interface gráfica no modo de *dashboard*. Foram utilizados os conhecimentos adquiridos durante a disciplina de sistemas operacionais, como aqueles relacionados a comandos, terminais e interfaces *shell*, CPU, memória, sistema de arquivos e processos. Este relatório apresenta as tecnologias utilizadas, bem como o resultado final obtido.

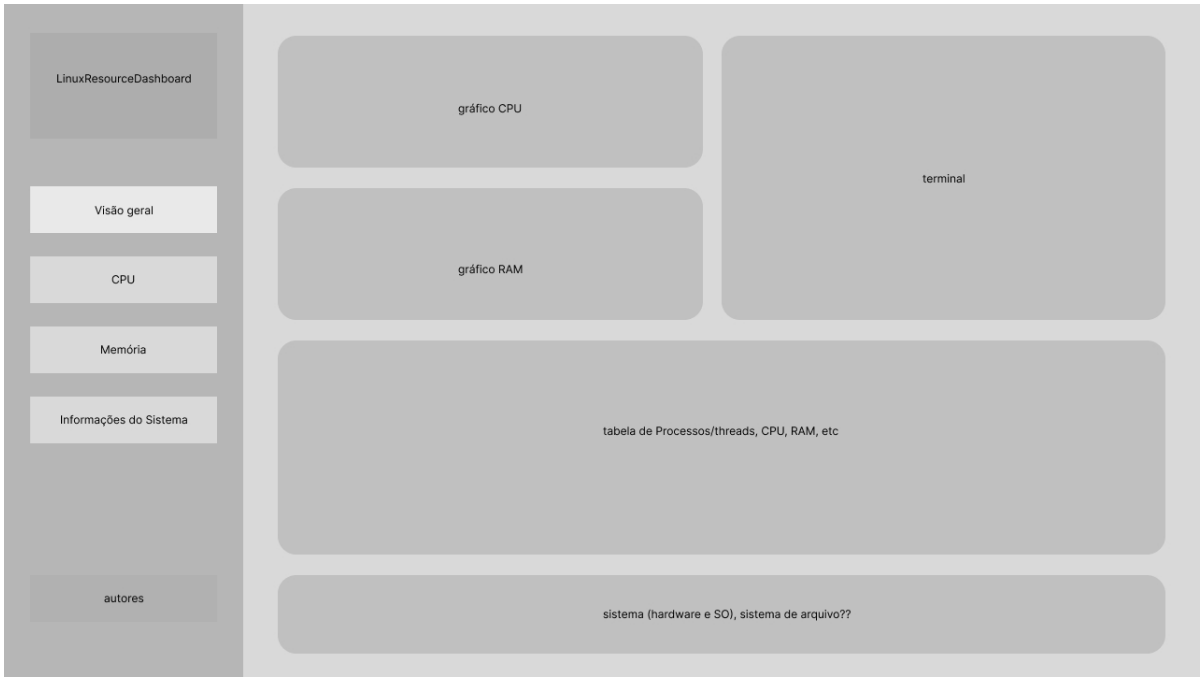
O código do *software LinuxResourcesDashboard* está disponível em https://github.com/gabteo/LinuxResourceDashboard.

2 TECNOLOGIAS UTILIZADAS

A linguagem escolhida para o desenvolvimento do *dashboard* foi o *Python*. As primeiras bibliotecas gráficas consideradas foram *Kivy* e *Tkinter*. Rapidamente, a primeira opação foi descartada. Contudo, após algumas experiências frustradas e limitações da biblioteca *Tkinter*, mesmo quando complementada por outra biblioteca, *CustomTkinter* (que permite expandir a personalização da interface), também foi reconsiderada, uma vez que estava claro que as dificuldades encontradas seriam muitas.

Assim, após a construção de um protótipo (ilustrado na figura **??**), o projeto foi reiniciado utilizando as bibliotecas *Dash* e *Plotly*, que permitem construir *dashboards* com muito mais rapidez, uma vez que foram construídas especificamente para este fim. Estas bibliotecas utilizam um navegador *web* para renderizar o *dashboard*.

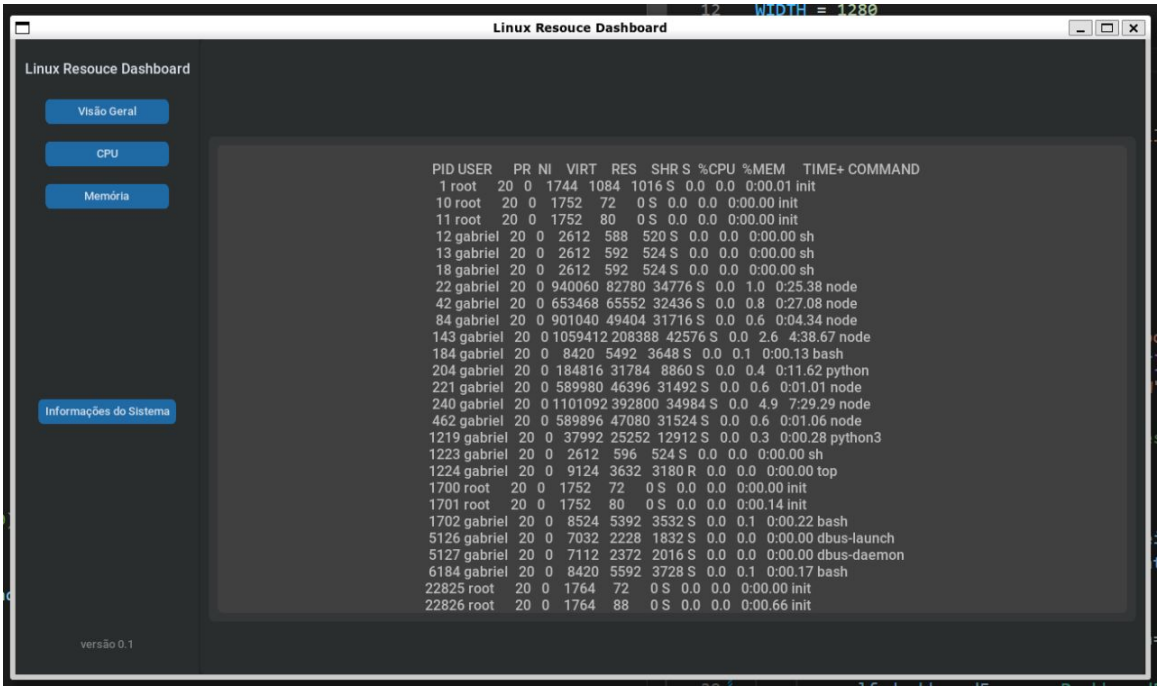
Figura 1 – Primeiro *Layout* planejado.



Quando o projeto foi reniciado com as novas bibliotecas, escolheu-se adotar uma visualização em uma única tela, em detrimento da antiga interface com guias, visto que concluiu-se , após pesquisas, que os *dashboards* costumam, por definição, concentrar diversas informações em um único painel.

Também no início do projeto, foi considerado armazenar os dados coletados do sistema em um banco de dados *SQLite*, com suas respectivas *timestamps*, para análise externa do desempenho do sistema a longo prazo, caso o usuário possua tais necessidades. Por isso, o código contém uma implementação parceialmente funcional deste recurso. Alguns testes iniciais foram realizados com sucesso, mas a funcionalidade não foi implementada completamente, devido às limitações de tempo para desenvolvimento. Entre os recursos testados estão a correta criação das tabelas de memória e uso da CPU (geral e por *cores*, de acordo com o processador instalado

Figura 2 – Primeiro protótipo com as bibliotecas *Tkinter* e *CustomTkinter*.



na máquina que executa o software).

Outro recurso muito utilizado durante o desenvolvimento foi a biblioteca *logging*, que permite gerar logs personalizados de execução de todos os comandos utilizados, bem como funções executadas, conforme a necessidade. A biblioteca foi de grande ajuda no desenvolvimento do *backend*, uma vez que permitiu acompanhar todo o funcionamento do programa, em termos de obtenção de dados, antes da conclusão da interface, acelerando o desenvolvimento e proporcionando muito mais controle (como definição de diferentes níveis de log - de erro a informações ou texto de *debug*) e personalização do que bibliotecas básicas que imprimem apenas na saída padrão do terminal.

3 INSTALAÇÃO E EXECUÇÃO

3.1 INSTALAÇÃO

É necessário ter instalado o *Python 3*.
No terminal, installe o *Dash* e o *Plotly*:

```
1 pip install dash
2 pip install plotly==5.11.0
```

3.2 EXECUÇÃO

Após clonar o repositório do github, acessar o diretório "dashApp"e executar o arquivo "dashboard.py"usando *Python 3*:

```
1 cd dashApp
2 python3 dashboard
```

No navegador de sua preferência (Google Chrome é recomendado), acesse o endereço "http://127.0.0.1:8050/".

Caso encontre problemas ao ao encerrar ou reiniciar o *dashboard* relacionados à porta 8050 que não foi fechado execute o comando abaixo:

```
1 lsof -i:8050 | grep python3
```

E encerre o processo cujo PID é o primeiro da lista retornada pelo comando anterior. Por exemplo, supondo que a saída seja:

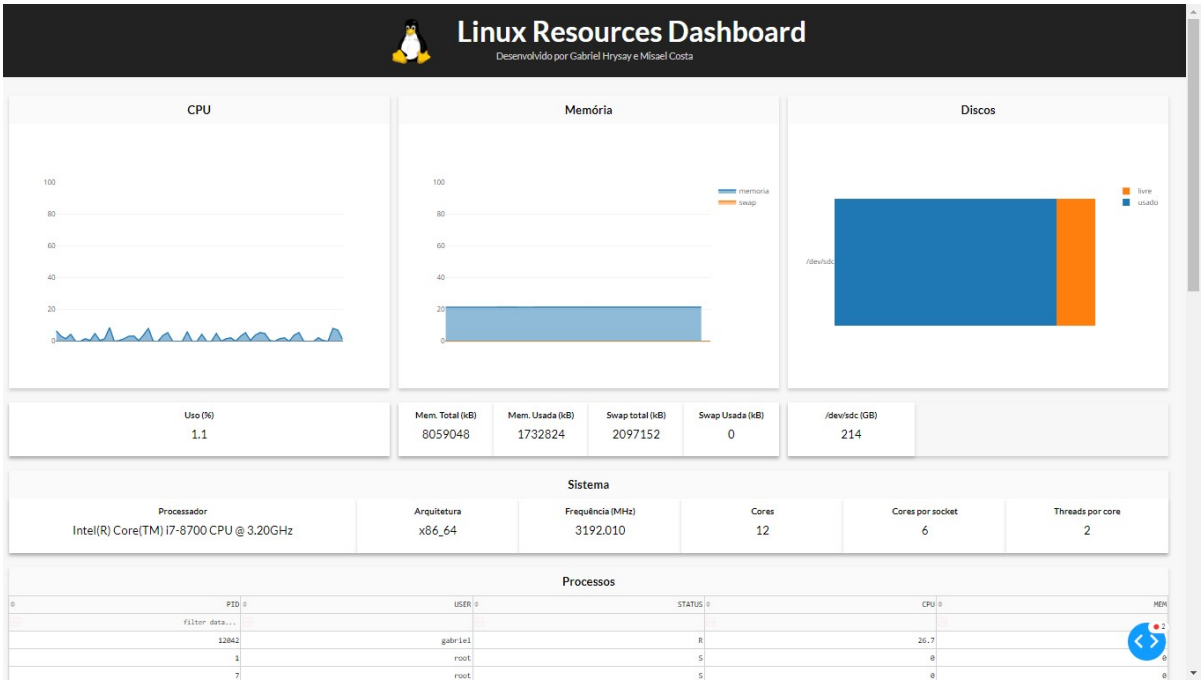
```
1 python3 11970 gabriel    4u  IPv4 1274422      0t0  TCP localhost:8050 (LISTEN)
2 python3 11970 gabriel    6u  IPv4 1274422      0t0  TCP localhost:8050 (LISTEN)
3 python3 12042 gabriel    4u  IPv4 1274422      0t0  TCP localhost:8050 (LISTEN)
4 python3 12042 gabriel    6u  IPv4 1274422      0t0  TCP localhost:8050 (LISTEN)
```

Execute:

```
1 kill -9 6840
```

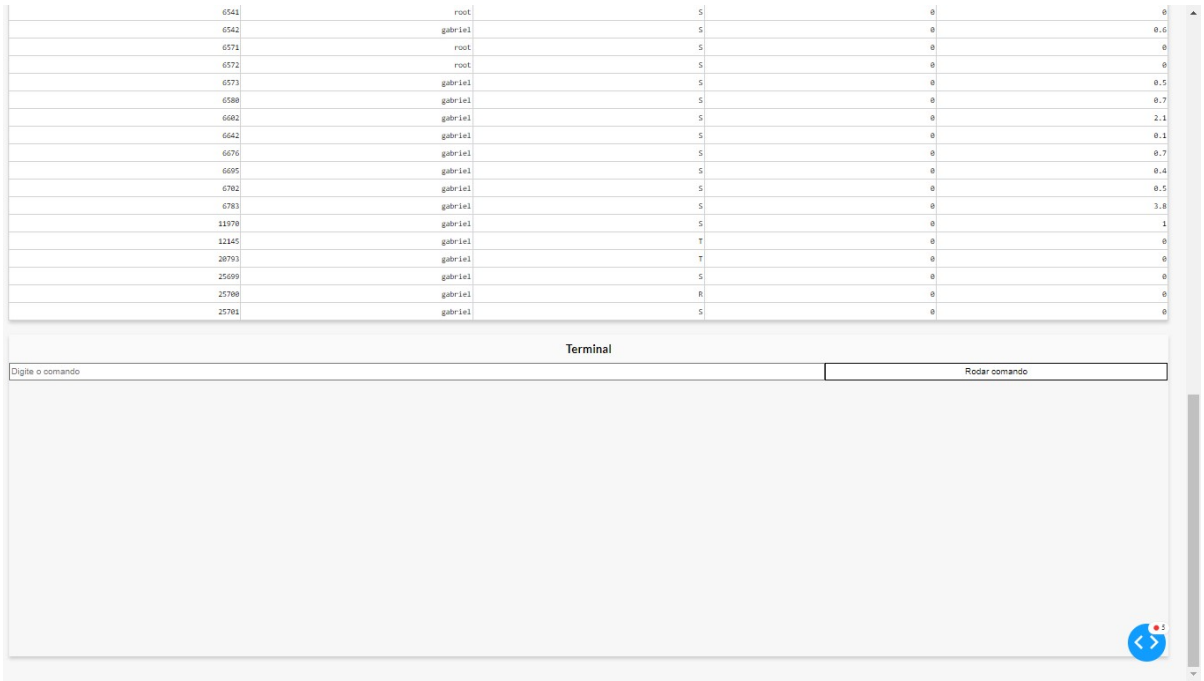
4 VISÃO GERAL E COMANDOS UTILIZADOS

Figura 3 – Tela inicial do *dashboard*.



Ao abrir o *dashboard*, vê-se a tela da figura 3. A interface tem áreas para CPU, memória, discos, informações do sistema e processos. Mais abaixo, rolando a tela, há um terminal, como mostra a figura 10

Figura 4 – Tela inicial do *dashboard*.

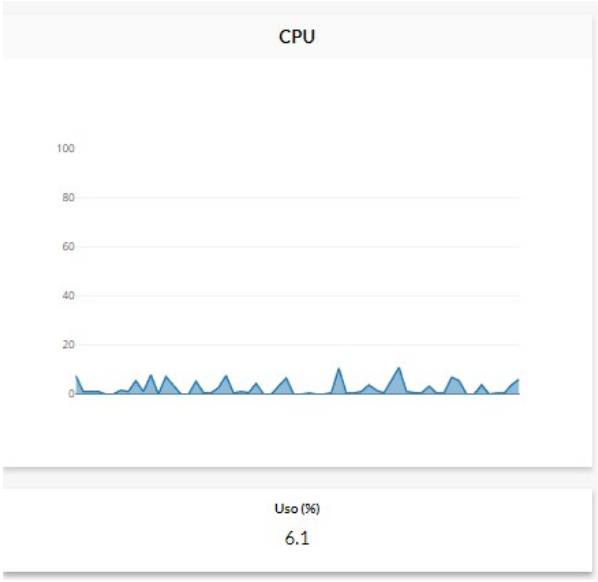


A interface foi elaborada com a intenção de apresentar os dados de modo intuitivo.

Ainda assim, nas próximas subseções, cada área da interface será apresentada individualmente, bem como os comandos usados no shell do Linux para obter as informações.

4.1 CPU

Figura 5 – Área de uso da CPU



A figura 5 mostra a área da CPU em detalhes. O gráfico mostra a evolução temporal do uso total percentual da CPU nos últimos 60 segundos, atualizado a cada segundo. O uso da CPU é obtido com o comando

```
1 top -bn1 | grep '%Cpu' |awk -F , '{print $4}' | awk '{print $1}'
```

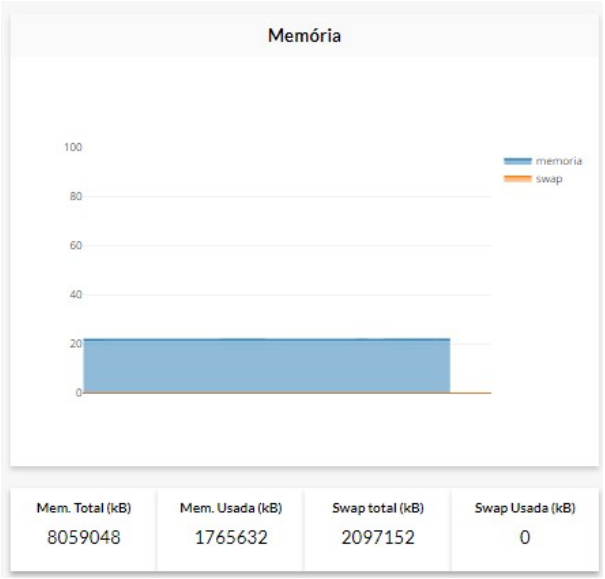
cuja saída é convertida para um valor float para exibição.

4.2 MEMÓRIA

A figura 6 mostra a área da Memória em detalhes. O gráfico mostra a evolução temporal dos usos totais percentuais da memória RAM e memória Swap nos últimos 60 segundos, atualizados a cada segundo. Os usos das memórias são obtidas com a seguinte função em *Python*:

```
1 def getMemStats(self):
2         dataToFetch = ["MemFree", "MemTotal", "MemAvailable",
3                        "SwapTotal", "SwapFree"]
4         self.statsDict = dict.fromkeys(dataToFetch)
5         for stat in dataToFetch:
6             cmd = "cat /proc/meminfo | grep {0} ".format(stat)
```

Figura 6 – Área de uso da Memória




```
7                cmd = cmd + "| awk '{print $2}'"
8
9                self.log.info(f"Executando comando: '{cmd}'")
10               output = int(self.execCmd(cmd))
11
12               #print(f"{stat}\t\t {output} kB")
13               self.statsDict[stat] = output
14
15           self.statsList = list(self.statsDict.values())
16
17           return self.statsList, self.statsDict
```

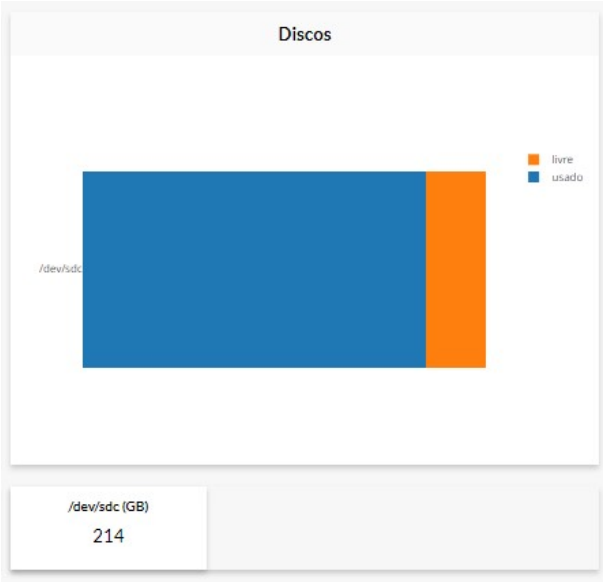
que retorna uma lista e um dicionário com as informações da lista *dataToFetch*. A função monta o comando para cada informação dessa lista. Por exemplo, para obter a memória total, a função monta o comando abaixo.

```
1 cat /proc/meminfo | grep MemTotal | awk '{print $2}'
```

4.3 DISCOS

A figura 7 mostra a área dos discos em detalhes. O gráfico mostra a ocupação percentual dos discos instalados no sistema. As informações são obtidas com o comando abaixo, manipulando a string da saída com o próprio *Python*.

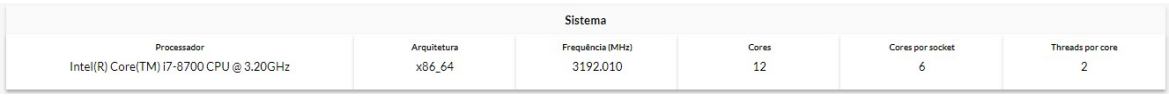
Figura 7 – Área de uso dos Discos




```
df -h --type=ext4
```

4.4 INFORMAÇÕES DO SISTEMA

Figura 8 – Área de informações do sistema



A área de informações do sistema apresenta as informações ilustradas na figura 8. Elas são obtidas através de uma função similar à apresentada na seção 4.2:

```
def getCpuInfo(self):
        dataToFetch = ['"Architecture"', '"CPU(s):"',
                '"Thread(s) per core"', '"Core(s) per socket"',
                '"Model name"', '"CPU MHz"']
        self.cpuInfoDict = dict.fromkeys(dataToFetch)
        for field in dataToFetch:
            cmd = "lscpu | grep {0} ".format(field)
            cmd = cmd + "| awk -F ':' '{print $2}'"

            self.log.info(f"Executando comando: '{cmd}'")
            output = self.execCmd(cmd)

```

```
13                print(f"{field}\t\t {output}")
14                self.cpuInfoDict[field] = output
15
16            self.cpuInfoList = list(self.cpuInfoDict.values())
17
18            return self.cpuInfoList, self.cpuInfoDict
```

4.5 PROCESSOS

Figura 9 – Área de Processos

Processos

PID	USER	STATUS	CPU	MEM
filter data...				
12042	gabriel	R	26.7	1.2
1	root	S	0	0
7	root	S	0	0
8	root	S	0	0
9	gabriel	S	0	0.1
71	root	S	0	0
1209	root	S	0	0
1210	root	S	0	0
1211	gabriel	S	0	0.1
6509	root	S	0	0
6510	root	S	0	0
6511	gabriel	S	0	0
6512	gabriel	S	0	0
6517	gabriel	S	0	0
6521	gabriel	S	0	0.9

A área de processos, mostrada em detalhes na figura 9, permite ordenar os processos pelos valores de qualquer uma das colunas apresentadas, clicando sobre as setas no cabeçalho da tabela. A ordenação pode ser crescente ou decrescente.

Também é possível filtrar os processos pelo conteúdo de uma coluna. Para mostrar apenas os processos do usuário *root*, por exemplo, basta clicar na cédula abaixo de "USER", na linha marcada com "filter data..."e digitar "root".

As informações exibidas são coletadas através do comando *top* do Linux:

```
1 top -n 1 -b | awk 'FNR > 7 { print }'
```

Os valores da *string* retornada pelo comando são separados em uma função do código.

4.6 TERMINAL

Enfim, a figura 10 mostra uma implementação simplificada do terminal. O comando digitado é executado e sua saída é exibida logo abaixo.

Figura 10 – Área de Terminal

5 CONCLUSÃO E TRABALHOS FUTUROS

O desenvolvimento deste software proporcionou grande aprendizado prático, tanto ao exigir a aplicação de diversos conceitos teóricos estudados quanto ao proporcionar interdisciplinariedade, à medida que fez-se necessário pesquisar e aprender novas tecnologias e bibliotecas até então desconhecidos pelos alunos envolvidos. A integração entre comandos para shell e uma linguagem de programação ilustraram o poder da combinação destas ferramentas. O trabalho também contribuiu para maior familiarização com comandos do terminal, não somente do Linux, uma vez que foram realizadas instalações via terminal e uma ferramenta de virtualização precisou ser instalada (*WSL: Windows Subsystem for Linux*). Consequentemente, a equipe ampliou conhecimentos sobre birtualização e divisão de recursos entre sistemas hospedeiro e hospedado.

O *software* desenvolvido poderia ser expandido com muitas funcionalidades, como introduzido na seção 2. Com um banco de dados armazenando os dados coletados no decorrer do tempo, o administrador de um servidor poderia consultar os dias e horários com uso mais intenso, por exemplo, e buscar implementar otimizações no sistema em execução nesse servidor para se adequar a essas condições. O programa também poderia ser configurado para emitir alertas para o usuário quando determinados recursos de um servidor ou máquina pessoal atingisse certa ocupação de armazenamento, ou quando o uso de sua memória RAM for frequentemente maior que um percentual, sugerindo um *upgrade* de um ou outro componte da máquina. Alguns campos com informações de *Hardware* e sistemas de arquivos poderiam ser acrescentados para aumentar a abrangência do *dashboard*, bem como gráficos independentes para cada núcleo da CPU.

Outras ideias de projetos também são frutos deste projeto, como a implementação de suporte para monitoramento de máquinas remotas (como os servidores mencionados acima), *dashboards* para monitoração de dispositivos IoT e de sistemas mocrocontrolados no geral. Com o conhecimento adquirido, também passa a ser possível gerar visualizações mais interessantes para dados tratados com *Python* em outras disciplinas e projetos pessoais. O mesmo ocorre com todo o conteúdo da disciplina, que tem aplicações notavelmente práticas e frequentes, como *threads* e *mutexes*, comunicação entre processos e chamadas de sistema (ambas usadas pela biblioteca *subprocess* ao executar um comando neste trabalho).