

**FACULDADE DE AMERICANA**  
**CIÊNCIA DA COMPUTAÇÃO**

**GABRIEL DA SILVA LIRA**

**NETSTATS**

Análise e visualização de dados em um provedor de redes

**AMERICANA**  
**2019**

**GABRIEL DA SILVA LIRA**

**NETSTATS**

Análise e visualização de dados em um provedor de redes

**Trabalho de conclusão de curso  
apresentado à Faculdade de Americana,  
como requisito parcial para obtenção do  
título de Bacharel em Ciência da  
Computação.**

**Orientador: Dr. Daives Arakem  
Bergamasco**

**AMERICANA  
2019**

**GABRIEL DA SILVA LIRA**

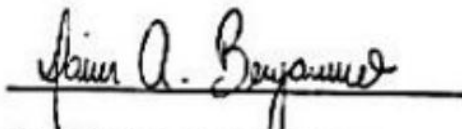
**NETSTATS**

**Análise e visualização de dados em um provedor de redes**

Trabalho de conclusão de curso  
apresentado à Faculdade de  
Americana, como requisito parcial para  
obtenção do título de Bacharel em  
Ciência da Computação.

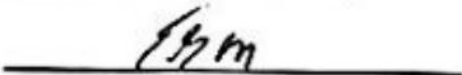
Data da aprovação: 26 / 11 / 2019

Banca Examinadora



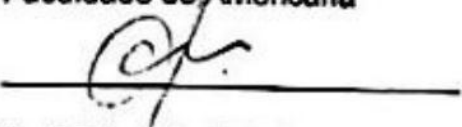
Prof. Daives Bergamasco

Faculdade de Americana



Prof. Edson Roberto Gaseta

Faculdade de Americana



Prof. Eduardo Sakai

Faculdade de Americana

## RESUMO

Empresas de telecomunicações utilizam diversas ferramentas para oferecer serviços de qualidade aos seus clientes, além da conexão física, que é o que se costuma enxergar, em telecomunicações são utilizadas APIs RESTful que são responsáveis por ser a “linha de chegada” de todos os protocolos que garantem acesso ao usuário final. O objetivo deste trabalho é trazer uma ferramenta capaz de analisar as respostas de um *webservice* de provisionamento utilizando a ciência de dados em busca de padrões para formulação de soluções para eventuais problemas que possam ocorrer em determinado recurso do sistema, essa ferramenta utiliza algoritmos para abstração de dados, identificação de padrões e concepção de gráficos em um sistema web simples e útil.

**Palavras-chave:** API; Webservice; Provisionamento.

## **ABSTRACT**

Telecommunications companies use various tools to provide quality services to their customers, in addition to the physical connection, which is commonly seen, in telecommunications RESTful APIs are used which are responsible for being the “finish line” of all protocols that guarantee end user access. The objective of this project is to bring a tool capable of analyzing the responses of a provisioning webservice using data science in search of patterns to formulate solutions for any problems that may occur in a given system resource. This tool uses algorithms for abstraction of data, pattern identification and graphic design in a simple and useful web system.

**Keywords:** API; Webservice; Provisioning.

## LISTA DE FIGURAS

Figura 1 - URL de acesso em uma API .....	13
Figura 2 - Funcionamento e um webservice.....	14
Figura 3 - Exemplo de arquivo CSV .....	16
Figura 4 - JSON do webservice .....	16
Figura 5 - Registro de operações .....	17
Figura 6 - Replace no Visual Studio Code .....	18
Figura 7 - Registro de operações classificado por tipo .....	18
Figura 8 - Arquivo CSV com as respostas do webservice .....	19
Figura 9 - Arquivos CSV como parâmetros .....	19
Figura 10 - Funções de Pandas .....	21
Figura 11 - Dataframe acessos por usuários .....	22
Figura 12 - Dataframe acessos por URL .....	22
Figura 13 - Dataframe status code .....	23
Figura 14 - FSN entre os registros .....	24
Figura 15 - Coleta de um FSN .....	25
Figura 16: Contagem de mensagens de erro ou sucesso .....	26
Figura 17: Calcular porcentagem .....	27
Figura 18: Gerar gráfico com Python .....	28
Figura 19: Gráfico de barras .....	29
Figura 20: Gráfico de pizza .....	30

## LISTA DE ABREVIações

API	Application Programming Interface.....
GPON	Gigabit Passive Optical Network.....
CSV	Comma-separated values.....
JSON	JavaScript Object Notation.....
XML	Extensible Markup Language.....
URI	Uniform Resource Identifier.....
URN	Uniform Resource Name.....
URL	Uniform Resource Locator.....
REST	Representational State Transfer.....
CRUD	Create, Read, Update and Delete.....
IPTV	Internet Protocol Television.....
VoIP	Voice Over Internet Protocol.....
HTTP	Hypertext Transfer Protocol.....
ONU	Optical Network Unit.....
FSAN	Full-Service Access Network.....
PNG	Portable Network Graphics.....
HTML	Hypertext Markup Language.....

## SUMÁRIO

<b>RESUMO .....</b>	<b>4</b>
<b>ABSTRACT.....</b>	<b>5</b>
<b>LISTA DE FIGURAS .....</b>	<b>6</b>
<b>LISTA DE ABREVIACÕES .....</b>	<b>7</b>
<b>1. INTRODUÇÃO .....</b>	<b>9</b>
1.1 ORGANIZAÇÃO DO TRABALHO.....	10
<b>2. REVISÃO BIBLIOGRÁFICA .....</b>	<b>11</b>
2.1 DADOS EM CIÊNCIA DA COMPUTAÇÃO .....	11
2.2 INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES .....	11
2.3 WEBSERVICE .....	12
2.4 ESTRUTURA DE SOFTWARE DA EMPRESA.....	13
<b>3. DESENVOLVIMENTO .....</b>	<b>15</b>
3.1 COLETA DE DADOS .....	15
3.2 ANÁLISE DE DADOS .....	20
3.2.1 Análise dos Logs de Acessos .....	20
3.2.2 Análise dos Logs de Operações .....	23
3.3 VIZUALIZAÇÃO DE DADOS .....	27
<b>4. RESULTADO .....</b>	<b>29</b>
<b>5. CONSIDERAÇÕES FINAIS.....</b>	<b>32</b>
5.1 TRABALHOS FUTUROS .....	32
<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>33</b>
<b>6. APÊNDICES.....</b>	<b>34</b>
6.1 APÊNDICE A – Código fonte do Netstats .....	34
6.2 APÊNDICE B – Licença de software.....	48



## 1. INTRODUÇÃO

A visualização de dados é uma forma de representação de dados que podem ser apresentados em forma de gráficos, tabelas e outras funções que dependem da ferramenta utilizada, esses dados podem ser obtidos através de uma base de dados e com uma linguagem de programação específica ou com módulos específicos para análise de dados, é possível trabalhar com essas informações de diversas formas, no Netstats foi utilizado Python, podemos escrever algoritmos para abstrair uma determinada parte da base de dados, fazer comparativos entre valores e utilizar estatística para gerar relatórios com a parte visual muito mais amigável e de fácil compreensão.

Nos próximos anos, no mundo dos negócios a visualização de dados será uma ferramenta essencial e indispensável de apoio à gestão. Segundo Simon Samuel em entrevista a SAS (Statistical Analysis System).

A visualização de dados mudará a maneira como nossos analistas trabalham com os dados. Espera-se que eles respondam aos problemas mais rapidamente e que busquem por mais insights, que olhem para os dados de maneira diferente, com mais imaginação. A visualização de dados vai promover uma exploração criativa dos dados.

Com o desenvolvimento de recursos que possibilitam a visualização de dados utilizando a ciência de dados, é possível a criação de várias ferramentas de apoio à gestão e de análise técnica em diversas áreas da Tecnologia da Informação, trazendo um recurso cada vez mais solicitado e que sem dúvida traz diversos resultados e benefícios para o operacional de qualquer empresa ou instituição.

Para chegar ao resultado que será transformado em gráficos, tabelas e outros tipos de visualização, é necessário fazer o tratamento de toda essa informação já que tudo vem em base de dados que costumam ter milhares de colunas e que na maioria dos casos são praticamente ilegíveis e levariam muito tempo para extrair qualquer tipo de informação que traga resultado.

A mineração é o processo de tratamento dos dados, neste processo ocorre a limpeza de dados, abstração e identificação de padrões que podem ser passíveis de

análise, a mineração de dados pode ou não ter algoritmos padrões, em muitos casos, como neste projeto, será necessário criar uma lógica específica para tratar a base de dados.

Aplicativos *web* são muito utilizados como ferramenta de apoio a profissionais de rede, o Netstats foi construído com o *Flask*, um *framework* minimalista de desenvolvimento *web* baseado nas bibliotecas *WSGI Werkzeug* e *Jinja2*, com o *Flask* foi possível escrever o *backend* em Python, já aproveitando a linguagem que possui diversas bibliotecas para análise de dados sendo possível manter tudo em apenas um *script*.

O objetivo deste trabalho é apresentar como a visualização de dados pode ser uma forma muito eficaz de gerar relatórios e análises mesmo através de bases de dados que contém milhares de linhas, como é o caso da base de dados utilizada neste trabalho, na qual chega a ultrapassar sessenta e oito mil linhas. Além de mostrar o poder de alguns *frameworks* da linguagem de programação Python.

## 1.1 ORGANIZAÇÃO DO TRABALHO

Os capítulos a seguir estão organizados da seguinte forma: no capítulo 2 é apresentado os conceitos para o entendimento do projeto criado. No capítulo 3 é apresentado todo o fluxo de desenvolvimento do projeto, está dividido nas etapas de coleta, análise e visualização de dados. No capítulo 4 é apresentado os resultados e como os dados podem ser visualizados. No capítulo 5 está a conclusão do projeto, instruções e expectativas para trabalhos futuros usando este trabalho. No capítulo 6 está os apêndices com o código fonte do projeto e a licença de software.

## 2. REVISÃO BIBLIOGRÁFICA

Os dados utilizados neste projeto foram coletados a partir de um provedor de internet da cidade de Sumaré no interior de São Paulo. Toda a análise foi feita a partir de uma base de dados armazenada em um arquivo no formato CSV, este arquivo contém o registro de requisições das APIs de provisionamento da empresa.

Os próximos tópicos deste capítulo irão explicar alguns conceitos importantes para o entendimento deste projeto.

### 2.1 DADOS EM CIÊNCIA DA COMPUTAÇÃO

Existe diferenças entre os conceitos de dado, informação e conhecimento, neste projeto foi realizado a transformação de dados em informação, não será utilizado o conceito de conhecimento dentro deste projeto, como será possível observar no decorrer dos próximos capítulos, apenas dados e informação será relevante para o Netstats.

Segundo Amaral (2016, p.3) os “dados são fatos coletados e normalmente armazenados. Informação é o dado analisado e com algum significado. O conhecimento é a informação interpretada, entendida e aplicada para um fim”. Amaral também cita os três tipos de dados, o dado não eletrônico, o dado analógico e o dado digital (AMARAL, 2016).

O Netstats trabalha apenas com o dado no formato digital, dados em formato digital são aqueles transmitidos por bits, ou seja, são armazenados apenas em zeros e uns.

### 2.2 INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES

API, acrônimo para *application programming interfaces*, traduzido para o português como interface de programação de aplicações, é um conjunto de solicitações padronizadas de software (ORENSTEIN, 2000).

Basicamente uma API é um conjunto de diversas funcionalidades fornecidas por um *software* que podem ser acessadas via código, desta maneira não é necessário utilizar todo um software, apenas a funcionalidade específica da API, através das APIs, os aplicativos podem se comunicar uns com os outros sem conhecimento ou intervenção dos usuários (CANALTECH, 2019). Em essência, a API de um programa define a maneira correta de um desenvolvedor solicitar serviços desse programa (ORENSTEIN, 2000).

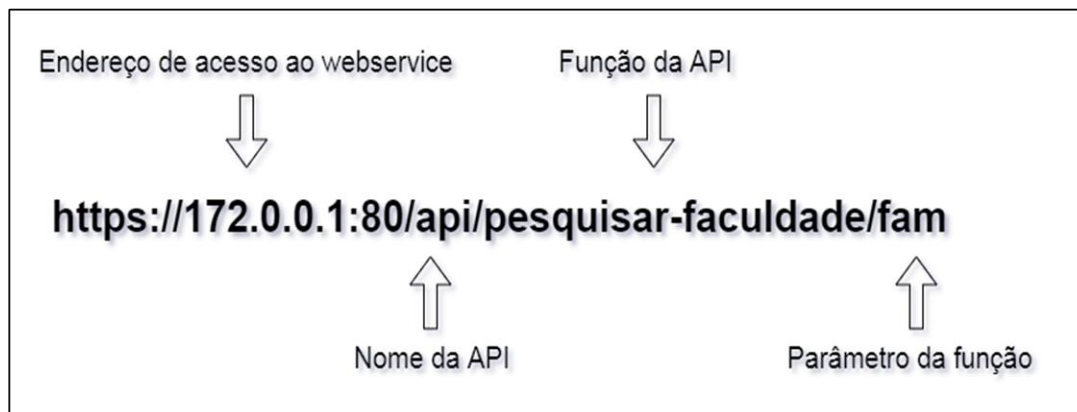
## 2.3 WEBSERVICE

*Webservice* é uma ferramenta que tem o objetivo de integrar várias aplicações com a finalidade de ser um ambiente comum operacional onde diversas APIs (mesmo que em linguagens e estruturas diferentes) possam se comunicar e compartilhar todos os seus recursos de maneira simples e padronizada, o retorno de todas as funções, independente da API, são traduzidos através do *webservice* em um formato padrão, geralmente JSON, XML, entre outros. O desenvolvimento e a utilização de um *webservice* em empresas de telecomunicações é fundamental, caso a empresa faça a opção de operar sua rede através de provisionamento, ou seja, configuração a distância, é necessário trabalhar com várias APIs de diferentes tecnologias, por isso um *webservice* facilita em muito e reduz tempo em operações diminuindo o ciclo operacional de um analista. O *webservice* importa todos os recursos das APIs para posteriormente ser acessado e devolver o retorno da função requisitada.

Um conceito importante para entender o funcionamento de um *webservice* é o termo URI, cujo acrônimo representa *Uniform Resource Identifier*, ou traduzido para a língua portuguesa como Identificador Uniforme de Recurso, este termo representa um conjunto de caracteres que são utilizados para identificar um recurso disponível na Internet. Um URI pode ser dividido em dois tipos, URN (*Uniform Resource Name* traduzido para o português como Nome de Recurso Uniforme) e URL (*Uniform Resource Locator* traduzido para o português como Localizador de Recurso Uniforme) que é o formato de acesso utilizado no *webservice* no qual foram coletados os dados neste projeto.

Na Figura 1 é possível ver como é feito o acesso de uma API através de um *webservice*.

Figura 1: URL de acesso em uma API (Endpoint)



Fonte: Gabriel da Silva Lira (2019)

O *webservice* utilizado neste projeto utiliza o conjunto de restrições denominado *Representational State Transfer (REST)*, em português Transferência Representacional de Estado, que é uma arquitetura de *software* baseado nas operações conhecidas como CRUD, acrônimo para *create*, *read*, *update*, *delete* (MARTIN, 1983), essas quatro definições são as quatro operações que um *webservice* realiza, a operação *create* é usada para adicionar elementos ao *webservice*, a operação *read* recupera dados e informações do *webservice*, a operação *update*, atualiza um elemento que já foi adicionado anteriormente modificando os seus parâmetros, e por fim a operação *delete* remove elementos do *webservice*.

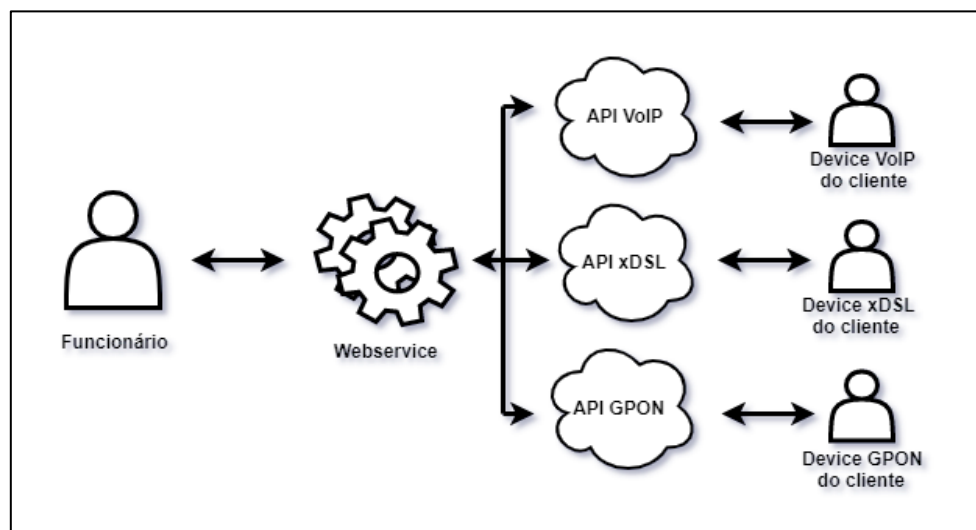
## 2.4 ESTRUTURA DE SOFTWARE DA EMPRESA

A empresa possui um *webservice*, a aplicação responsável por gerenciar e integrar todas as APIs de provisionamento da empresa, basicamente cada tecnologia, como por exemplo, GPON, IPTV e VoIP, possui uma API própria que é associada ao *software* do fabricante de equipamentos que a empresa usa para oferecer cada tipo de serviço a seus clientes, essas APIs funcionam definindo parâmetros e

configurações para cada cliente e podem ser alterados por funcionários da empresa para definir esses parâmetros, todo o processo de configuração é feito a distância.

O processo de configuração de um equipamento de rede remotamente é chamado de provisionamento, na Figura 2 é possível ver como funciona um *webservice* composto por APIs de provisionamento.

Figura 2: Funcionamento de um webservice



Fonte: Gabriel da Silva Lira (2019)

### 3. DESENVOLVIMENTO

#### 3.1 COLETA DE DADOS

Primero é importante ressaltar que os dados coletados neste projeto não trazem nenhuma informação que comprometa a empresa ou exponha qualquer tipo de informação sigilosa, nem da empresa e principalmente dos clientes, nenhuma informação de qualquer cliente é coletada ou exposta, o sistema apenas mantém o registro de chamadas e o status da operação solicitada das APIs de provisionamento da empresa, nada foi coletado ou utilizado de forma irresponsável. A ética deve estar sempre aliada a qualquer tipo de pesquisa não podendo ultrapassar os limites da privacidade de qualquer indivíduo.

Uma das chamadas do *webservice* utilizado neste projeto retorna *logs* do sistema, *logs* são os registros de requisições do protocolo HTTP, cada requisição de qualquer usuário de qualquer sistema que está acessando o *webservice* irá ser registrado. Existem duas funções do *webservice* que retornam *logs*, uma retorna *logs* de acesso, que traz informações de acesso das APIs ao *webservice*, tais como API que acessou, horários e código de *status*, já outra retorna os *logs* de operações, que traz informações das operações requisitadas, código de *status*, entre outros dados. Essas chamadas foram acessadas por terminal e todos os dados coletados foram salvos em um arquivo de extensão CSV.

CSV é um formato de arquivo regulamentado pelo RFC 4180, basicamente composto por colunas separadas por vírgula, o módulo da linguagem Python que foi utilizado para coletar os dados, identifica a primeira linha automaticamente como o título de cada coluna. Na Figura 3 é possível ver a estrutura de um arquivo CSV, importante lembrar que a estrutura deste tipo de arquivo apenas define colunas separadas por vírgula, utilizar a primeira linha da coluna como título é um tipo de formatação válido apenas para o módulo da linguagem Python utilizado neste projeto.



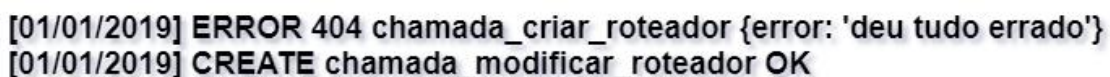


porém desta forma a leitura, análise e interpretação deste conjunto de dados é praticamente impossível.

Estes dois conjuntos de dados, que são os *logs* de acesso e *logs* de operações, servirão como parâmetros para as funções do Netstats. O primeiro passo antes de usar estes dados como parâmetro, é deixar o arquivo legível e tratável por uma linguagem de programação. Em programação, basicamente as operações se baseiam em laços e condições, é necessário que estes arquivos tenham a estrutura básica para trabalhar com essas operações.

Observe a Figura 5.

Figura 5: Registro de operações

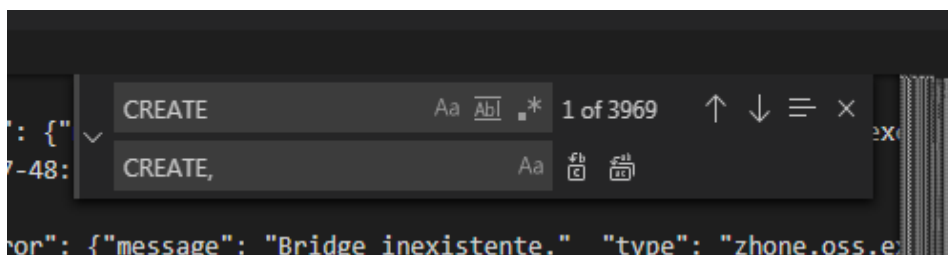


```
[01/01/2019] ERROR 404 chamada_criar_rotador {error: 'deu tudo errado'}  
[01/01/2019] CREATE chamada_modificar_rotador OK
```

Fonte: Gabriel da Silva Lira (2019)

Na Figura 5 é possível ver um exemplo aproximado de duas linhas que representam dois registros das operações. Primeiro foi necessário uma análise e ajustes manuais, observando a Figura 5 é possível notar que existe um padrão em cada linha mesmo que em operação e resultado diferentes, a primeira informação que consta em cada linha é a data, seguido pelo tipo de operação e finalizando com a mensagem de retorno, neste momento já é possível fazer uma abstração antes mesmo de entrar no sistema Netstats, analisando a linha é possível visualizar que a data sempre é finalizada com um colchete, com um editor de texto qualquer, no caso foi utilizado o *Visual Studio Code*, bastou apenas selecionar todos os colchetes invertidos e substituir pelo próprio colchete com uma vírgula a seguir, após a data vem o tipo de operação, que no caso são apenas quatro, “*CREATE*”, “*DELETE*”, “*PUT*” e “*GET*”, assim como na data bastou apenas selecionar todas estas operações no editor de texto e em seguida substituir pela própria operação com uma vírgula ao final, observe o exemplo na FIGURA 6 de como fazer esta substituição.

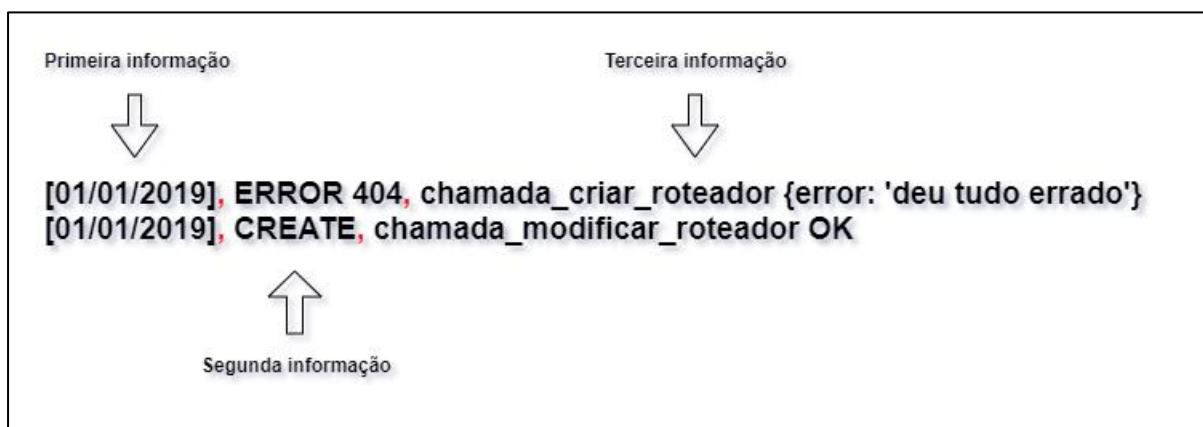
Figura 6: Replace no Visual Studio Code



Fonte: Gabriel da Silva Lira (2019)

Com isso, a linha inteira foi classificada por tipo de informação, e por fim o exemplo da Figura 5 ficara assim como na Figura 7.

Figura 7: Registro de operações classificado por tipo



Fonte: Gabriel da Silva Lira (2019)

Desta forma cria-se colunas que representam cada tipo de informação, podendo assim ser manipulado de forma abstraída por uma linguagem de programação.

Finalizando a etapa da limpeza de dados, foi necessário apenas nomear cada coluna na primeira linha, na Figura 8 está o arquivo CSV pronto para ser usado como parâmetro das funções do Netstats, este exemplo contém o arquivo com os *logs* de operações, foi feito as mesmas etapas para os *logs* de acesso, porém o arquivo possui algumas colunas a mais.

Figura 8: Arquivo CSV com as respostas do webservice

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	data,tipo,operacao																	
2	[20/Mar/2019:07:10:41],ERROR 404, DELETE onu_delete: {"error": {"message": "Onu nao provisionada." "type": "zhone.oss.exceptions.AppException"}}: fsan 0388AECE																	
3	[20/Mar/2019:07:11:41],CREATE, smart fsan 038AD969 ssp 1-14-7-48: delete slot_olt_port_delete																	
4	[20/Mar/2019:07:11:56],CREATE, smart fsan 038AD969: create																	
5	[20/Mar/2019:07:12:58],ERROR 404, onu_bridge_path_list: {"error": {"message": "Bridge inexistente." "type": "zhone.oss.exceptions.AppException"}}: fsan 0379A185 vlan video																	
6	[20/Mar/2019:07:14:29],CREATE, smart onu_home_create OK fsan 038AD969 ip [REDACTED]																	
7	[20/Mar/2019:07:27:56],ERROR 404, sysname ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
8	[20/Mar/2019:07:27:57],ERROR 404, session_uptime ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
9	[20/Mar/2019:07:27:57],ERROR 404, uptime ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
10	[20/Mar/2019:07:27:57],ERROR 404, channel_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
11	[20/Mar/2019:07:27:57],ERROR 404, color ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
12	[20/Mar/2019:07:27:58],ERROR 404, mac_ap ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
13	[20/Mar/2019:07:27:58],ERROR 404, interface_lan_status_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
14	[20/Mar/2019:07:27:58],ERROR 404, power_level_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
15	[20/Mar/2019:07:27:58],ERROR 404, status_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
16	[20/Mar/2019:07:27:58],ERROR 404, mac_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
17	[20/Mar/2019:07:27:58],ERROR 404, jitter_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
18	[20/Mar/2019:07:28:03],ERROR 404, power_level_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
19	[20/Mar/2019:07:28:03],ERROR 404, jitter_sm ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	
20	[20/Mar/2019:07:28:39],ERROR 404, uptime ip [REDACTED]: {"error": {"message": "No SNMP response received before timeout" "type": "wifioss.wifioss.WifiException"}}}																	

Fonte: Gabriel da Silva Lira (2019)

Com o arquivo pronto para ser analisado, agora chega a etapa de coleta através do Netstats, conforme descrito anteriormente, estes arquivos servirão como parâmetro das funções do Netstats, para isso foi necessário utilizar o Pandas.

Pandas é um módulo da linguagem Python, ou biblioteca como seria chamado em outras linguagens, se trata de um projeto *open source* criado por Wes McKinney, um desenvolvedor de *software* e empresário norte-americano. Pandas foi criado com o intuito de prover uma ferramenta capaz de gerar análises de forma simples em qualquer conjunto de dados utilizando a linguagem de programação Python, o módulo Pandas foi utilizado tanto na coleta quanto na análise e visualização de dados neste projeto. Observe a Figura 9.

Figura 9: Arquivos CSV como parâmetros

```

6
7 import pandas as pd
8
9
10 logs = pd.read_csv(r'static/websvc_access.csv')
11 operacao = pd.read_csv(r'static/websvc_error1.csv')
12
13
14 class Netstats:
15
16     def __init__(self):
17         self.access = Access(logs)
18         self.fsan = Fsan(operacao)
19         self.lista_data = ListaDataframe(operacao)
20         self.error = Error(operacao)
21         self.estatisticas = EstatisticasGerais(operacao)
22
23

```

Fonte: Gabriel da Silva Lira (2019)

Finalizando a etapa de coleta de dados, na Figura 9 é possível ver como os arquivos de *logs* foram utilizados como parâmetros para as funções do Netstats, observe que primeiro é feito a importação do módulo Pandas, não seria obrigatório, mas “pd” foi definido como representação do módulo. Na classe Netstats está contido uma função inicializadora, não é necessário entrar em detalhes sobre o desenvolvimento agora, mas é possível notar que esta função inicializadora importa as outras funções do Netstats que estão em outros arquivos, todas estas funções necessitam de um conjunto de dados como parâmetro para a execução, após importar o módulo Pandas, foi necessário utilizar a função deste módulo chamada “*read\_csv*” para a leitura do arquivo de extensão CSV, já definindo dentro de uma variável, as variáveis “logs” e “operacao” são responsáveis por armazenar os dois conjuntos de dados, logo após, na função inicializadora, é feito a definição destas duas variáveis como parâmetro para as funções.

## 3.2 ANÁLISE DE DADOS

Com os arquivos prontos e já definidos como parâmetros para o Netstats, neste tópico será possível observar como as os gráficos do Netstats são gerados.

### 3.2.1 Análise dos Logs de Acessos

Como já descrito anteriormente na etapa de coleta de dados, agora todos os dados pertencentes ao arquivo de *logs* de acesso foram lidos e incluídos como um argumento de uma classe. Para os *logs* de acesso foram definidas dentro de uma classe nomeada “Access” três abstrações para análise, acessos por usuários, acessos por URL e o código de status. O arquivo de *logs* de acesso está separado em quatro colunas da qual foram analisadas três, “usuários”, “URL” e “status code”, cada uma dessas colunas foram analisadas separadamente.

Basicamente a análise desta parte da base de dados é feita pela constatação da frequência, ou seja, o número de ocorrências para cada tipo de abstração. Conforme é possível observar na Figura 9, a partir do momento em que foi utilizado o módulo Pandas para capturar os arquivos como argumento, é possível acessar as

várias funções deste módulo, por exemplo, a partir da definição de “*logs = pandas.read\_csv(arquivo)*” será possível acessar qualquer função de Pandas usando “*logs.função()*”. Para as três abstrações na análise de acessos, foi necessário usar a função “*value\_counts()*”, que é responsável por agrupar de acordo com a frequência dados que possuem o mesmo valor.

Observe o exemplo na Figura 10 de como foram usadas as funções de Pandas em uma das três abstrações.

Figura 10: Funções de Pandas

```
21  
22     acessos_por_usuario = self.logs.usuario.value_counts().to_frame().reset_index()  
23     acessos_por_usuario.columns = ['Usuário', 'Acessos']  
24
```

Fonte: Gabriel da Silva Lira (2019)

No exemplo da Figura 10 foi feito o acesso apenas da coluna “*usuario*”, em seguida foi necessário utilizar três funções de Pandas, a função “*value\_counts()*” como já descrito anteriormente é responsável por agrupar frequência de valores, a função “*to\_frame()*” que transforma esta variável em uma *dataframe*, em softwares relacionados a estatísticas *dataframes* representam tabelas de dados, e também foi utilizado a função “*reset\_index()*” que apenas remove o índice do *dataframe*. Estes *dataframes* possuem duas colunas, uma com o dado específico e outra com a frequência em que este dado aparece, Pandas permite renomear cada coluna, é possível visualizar isso observando a linha vinte e no exemplo da Figura 10.

A primeira função da classe “*Access*” é analisar a quantidade de acessos por usuário, com base no conceito de *webservice* conforme descrito no capítulo 2, usuários representam APIs que acessaram o *webservice*, importante não confundir com usuário humano, neste caso a função “*value\_counts()*” irá agrupar a frequência em que as APIs acessam o *webservice*. O Netstats armazena um *dataframe* semelhante ao exemplo que pode ser observado na Figura 11 a seguir.

Figura 11: Dataframe acessos por usuários

Usuario	Acessos
0	sis 33581
1	ossclient 29911
2	zabbix 4463
3	netcore 74
4	weboss 68
5	- 39
6	diego 10
7	bkptelnet 3

Fonte: Gabriel da Silva Lira (2019)

Com isto já é possível extrair a primeira informação destes dados, é possível saber quais são as APIs mais acessadas e as menos acessadas.

A segunda função da classe “Access” é analisar a quantidade de acessos por URL, como já descrito anteriormente no capítulo 2, o acesso as funções do *webservice* são feitos a partir de URLs específicas, neste caso a função “*value\_counts()*” irá agrupar a frequência de acessos para cada URL. Na Figura 12 é possível ver um exemplo em *dataframe*.

Figura 12: Dataframe acessos por URL

url	acessos
0	/oss/gpon/onu_status 12661
1	/oss/gpon/check_device_ip 9183
2	/oss/gpon/omci_onu_status 8200
3	/snmponu/interface_traffic_in_wan/fsan 7306
4	/snmponu/interface_traffic_out_wan/fsan 4856

Fonte: Gabriel da Silva Lira (2019)

Esta informação pode ser considerada a mais importante extraída dos *logs* de acessos, obtendo a frequência de acessos por URL é possível saber quais são as funções do *webservice* que são mais utilizados.

A terceira função da classe “Access” é analisar a quantidade de cada código de status HTTP, conforme descrito anteriormente no capítulo 2, cada resposta de uma requisição HTTP possui um código de status, neste caso a função “*value\_couts()*” irá agrupar a frequência para cada código de status. É possível verificar no *dataframe* de exemplo na Figura 13.

Figura 13: Dataframe status code

Code	Frequencia
0	200 64643
1	404 1952
2	204 582
3	403 36
4	500 15
5	401 3
6	400 3

Fonte: Gabriel da Silva Lira (2019)

Esta informação pode ser considerada a menos importante entre os dados obtidos no arquivo de *logs* de acessos, porém é interessante observar a quantidade de informações que é possível extrair utilizando o módulo Pandas.

### 3.2.2 Análise dos Logs de Operações

O arquivo de *log* de operações possui o registro com as respostas para as APIs de todas as tecnologias, porém o Netstats analisa apenas operações relacionadas ao protocolo GPON, isso se dá pelo fato de que analisando o arquivo CSV previamente e conhecendo o funcionamento do *webservice* e das tecnologias no quesito operacional, só é possível identificar todo um processo realizado em GPON por ser a única tecnologia que possui uma identificação que pode ser constatada em todas as operações adicionais.

Para entender melhor sobre a tal identificação, em GPON, o equipamento receptor de sinal responsável por converter o sinal óptico em sinal elétrico na tecnologia GPON é a ONU (*Optical Network Unit*), a ONU possui um número identificador que é chamado de FSAN (*Full Service Access Network*), homônimo a organização que padroniza produtos e serviços relacionados a redes de computadores. Um FSAN é composto por 12 algarismos em hexadecimal e serve como uma identificação única para cada equipamento, no *webservice* cada resposta relacionada a um plano de GPON foi programada para atribuir o número do FSAN na resposta.

Observe a Figura 14.

Figura 14: FSAN entre os registros

```

1 data,tipo,operacao
2 [20/Mar/2019:07:10:41],ERROR 404, DELETE onu_delete: {"error": {"message": "Onu nao provisionada." "type": "zhone.oss.ex
3 [20/Mar/2019:07:11:41],CREATE, smart fsan 038AD969 ssp 1-14-7-48: delete slot_olt_port_delete
4 [20/Mar/2019:07:11:56],CREATE, smart fsan 038AD969: create
5 [20/Mar/2019:07:12:58],ERROR 404, onu_bridge_path_list: {"error": {"message": "Bridge inexistente." "type": "zhone.oss.e
6 [20/Mar/2019:07:14:29],CREATE, smart onu home create: K fsan 038AD969 ip 10.10.10.10
7 [20/Mar/2019:07:27:56],ERROR 404, sysname ip 10.10.10.10: {"error": {"message": "No SNMP response received before timeout
8 [20/Mar/2019:07:27:57],ERROR 404, session uptime ip 10.10.10.10: {"error": {"message": "No SNMP response received before
9 [20/Mar/2019:07:27:57],ERROR 404, uptime ip 10.10.10.10: {"error": {"message": "No SNMP response received before timeout
10 [20/Mar/2019:07:27:57],ERROR 404, channel sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before tim
11 [20/Mar/2019:07:27:57],ERROR 404, color ip 10.10.10.10: {"error": {"message": "No SNMP response received before timeout
12 [20/Mar/2019:07:27:57],ERROR 404, mac ap ip 10.10.10.10: {"error": {"message": "No SNMP response received before timeout
13 [20/Mar/2019:07:27:58],ERROR 404, interface lan status sm ip 10.10.10.10: {"error": {"message": "No SNMP response receiv
14 [20/Mar/2019:07:27:58],ERROR 404, power_level sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before
15 [20/Mar/2019:07:27:58],ERROR 404, status sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before time
16 [20/Mar/2019:07:27:58],ERROR 404, mac sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before timeout
17 [20/Mar/2019:07:27:58],ERROR 404, jitter sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before time
18 [20/Mar/2019:07:28:03],ERROR 404, power_level sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before
19 [20/Mar/2019:07:28:03],ERROR 404, jitter sm ip 10.10.10.10: {"error": {"message": "No SNMP response received before time

```

Fonte: Gabriel da Silva Lira (2019)

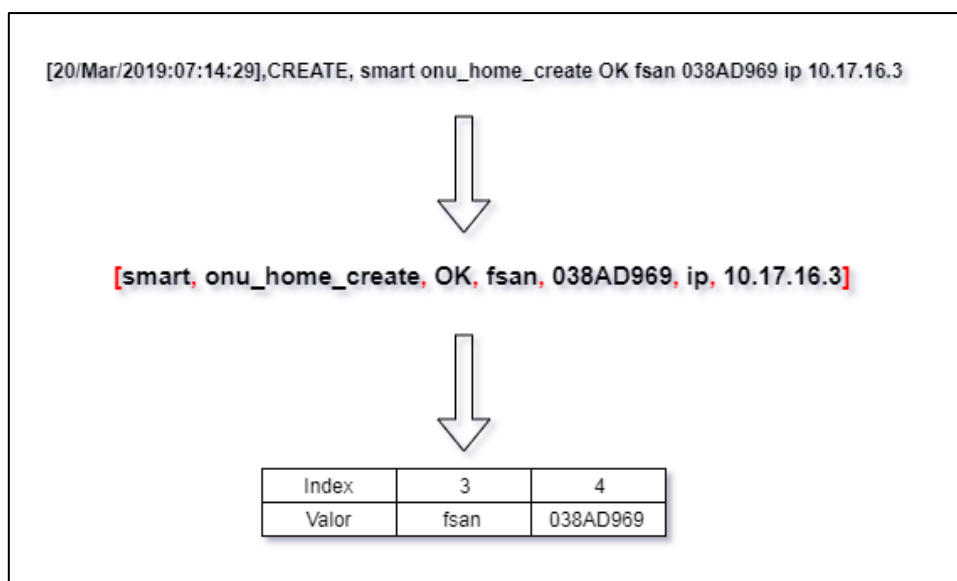
Na Figura 14 é possível observar o número do FSAN como componente das mensagens relacionadas a planos GPON, com isso, mesmo que as linhas do arquivo estejam completamente desordenadas, a partir deste identificador, o primeiro objetivo para extrair alguma informação destes dados é fazer o sistema identificar todos os FSANs do arquivo e após isso, agrupar todas as operações relacionadas a este FSAN.

A classe “Error” é responsável por gerar as informações do arquivo de logs de operações, antes da utilização desta classe foi necessário a criação de três classes que tem por objetivo geral identificar processos separados.

No primeiro momento, a classe “Fsan” coleta apenas a coluna com as mensagens de retorno e cria uma lista separada. Após isso, como é possível observar na Figura 14, o número do FSAN aparece sempre após a palavra “fsan” dentro da mensagem de retorno, e usando essa lógica o Netstats coleta todas as mensagens que possuem a palavra “fsan” e transforma a mensagem que é uma *string* em uma lista, cada palavra é transformada em um elemento e uma posição da lista, uma lista é um conjunto ordenado de valores, em que cada valor é identificado por um índice (ELKNER, DOWNEY e MEYERS, 2002, p. 81). Após a criação da lista, é possível conseguir o valor do FSAN localizando a palavra “fsan” dentro da lista e adicionar o próximo elemento da lista dentro de outra lista separada, na Figura 15 é possível ver como o sistema faz esta operação.



Figura 15: Coleta de um FSAN



Fonte: Gabriel da Silva Lira (2019)

A classe “*Fsan*” retorna todos os números de FSAN em uma lista, com isto, a classe “*ListaDataframe*” utiliza o retorno da classe “*Fsan*” para verificar em todo o arquivo em qual mensagem este identificador aparece, isto serve para agrupar todas as mensagens que estejam relacionados a um determinado número de FSAN, o retorno da classe “*ListaDataframe*” é uma lista que contém em cada posição um *dataframe* com todas as operações relacionadas a este número de FSAN, e o título do *dataframe* é o próprio número de FSAN, isso é ideal para a localização deste grupo de operações.

A classe “*ListaMensagens*” verifica o retorno da classe “*ListaDataframe*” e cria uma lista apenas com a última mensagem de cada operação. Importante frisar que a criação da classe “*ListaMensagens*” justifica as outras classes anteriores, o objetivo que foi determinado para analisar neste arquivo de *logs* de operações é identificar todo um processo e não apenas cada chamada individualmente, quando um usuário de um sistema que consome este *webservice* realiza qualquer uma de suas tarefas, como por exemplo, criar um plano GPON, por traz desta chamada do *webservice* são realizadas diversas outras operações, como por exemplo, criar serviço VoIP e IPTV e associar a ONU, além de que uma destas chamadas adicionais podem falhar e ser retomadas dentro do mesmo processo, neste caso mesmo com a falha individual, o processo em geral será considerado como sucesso, com isto é possível isolar cada tarefa de um funcionário e analisar o desempenho operacional da empresa.

E por fim, assim como na classe “Access”, a classe “Error” que posteriormente será utilizada para gerar a visualização da informação, é responsável por gerar três abstrações na base de dados, o percentual de sucesso e erro de todas as operações, a quantidade de operações que obtiveram sucesso e a quantidade de operações que obtiveram erros. Para todas as três abstrações definidas nesta classe, foi necessário utilizar o retorno da classe “ListaMensagens”, como descrito anteriormente esta classe retorna a última mensagem para cada grupo de operações.

A primeira função da classe “Error” é verificar o percentual de sucesso e erros para cada grupo de operações, para isto basta percorrer toda a lista gerada pela classe “ListaMensagens” e identificar palavras que pertence a uma mensagem de sucesso ou de erro, com um laço de repetição, basta apenas definir uma variável que serve como contador para quantificar as duas possibilidades. Para fazer isso em programação, basta criar um laço de repetição e criar uma condição em relação as mensagens de erro e sucesso, para cada mensagem criada o laço de repetição adiciona mais um número no valor da contagem, é possível ver este processo na Figura 16.

Figura 16: Contagem de mensagens de erro ou sucesso

```

22
23     contador_creates = 0
24     for item in ultima_msg:
25         if 'OK' in item or 'onu_business_create' in item or 'voip_create:' in item or '\
26             onu_delete: fsan' in item:
27             contador_creates += 1
28
29     contador_error = 0
30     for item in ultima_msg:
31         if 'error' in item:
32             contador_error += 1
33
34     data_ocorrencia = {
35         'Resultado': ['Sucesso', 'Erro'],
36         'FSANs': [contador_creates, contador_error]
37     }
38

```

Fonte: Gabriel da Silva Lira (2019)

Como é possível observar na Figura 16, após os laços de repetição, os respectivos contadores são adicionados em um dicionário, após isso para a criação do gráfico, que será demonstrado no capítulo 3.3, foi necessário transformar esses valores em porcentagem, é possível observar a lógica utilizada para calcular o percentual na Figura 17 a seguir.

Figura 17: Calcular porcentagem

```
56  
57     resultado_sucesso = (data_ocorrencia['FSANs'][0]*100)/len(ultima_msg)  
58     resultado_error = (data_ocorrencia['FSANs'][1]*100)/len(ultima_msg)  
59
```

Fonte: Gabriel da Silva Lira (2019)

Como é possível observar na Figura 17, o percentual é igual ao número de operações de erro ou sucesso multiplicado por cem e dividido pelo número total de operações.

As outras duas abstrações da classe “*Error*” retorna a frequência para cada tipo de operação, isso é feito contando a frequência em que algumas mensagens específicas aparecem, por exemplo, para todas mensagens que obtiveram sucesso, entre elas existem operações para criar serviços VoIP, para verificar a frequência para esta chamada específica basta apenas em um laço, verificar se existe “*voip\_create*” entre a mensagem.

### 3.3 VIZUALISAÇÃO DE DADOS

Para a visualização de dados foram utilizados dois módulos, ou biblioteca como são chamados em outras linguagens de programação. Foi necessário utilizar Matplotlib, uma biblioteca de plotagem 2D do Python que produz números de qualidade de publicação em vários formatos de cópia impressa e ambientes interativos entre plataformas (MATPLOTLIB, 2012), e também Seaborn que é uma biblioteca de visualização de dados Python baseada no Matplotlib (SEABORN, 2012).

São duas as classes responsáveis por gerar os gráficos, “*Access*” e “*Error*”, para os *logs* de acessos e *logs* de operações respectivamente. Tanto para os *logs* de acessos e os *logs* de operações, foram gerados os gráficos para as abstrações que foram definidas e descritas no capítulo 3.2, para gerar um gráfico foi usado os dicionários criados nessas abstrações.

Observe a Figura 18.

Figura 18: Gerar gráfico com Python

```
21  
22     acessos_por_usuario = self.logs.usuario.value_counts().to_frame().reset_index()  
23     acessos_por_usuario.columns = ['Usuário', 'Acessos']  
24  
25     plt.figure()  
26     plt.tight_layout()  
27     graf_cod = sns.barplot(x=acessos_por_usuario['Usuário'], y=acessos_por_usuario['Acessos'])  
28     graph = graf_cod.get_figure()  
29     graph.savefig('static/acessos_por_usuario.png')  
30
```

Fonte: Gabriel da Silva Lira (2019)

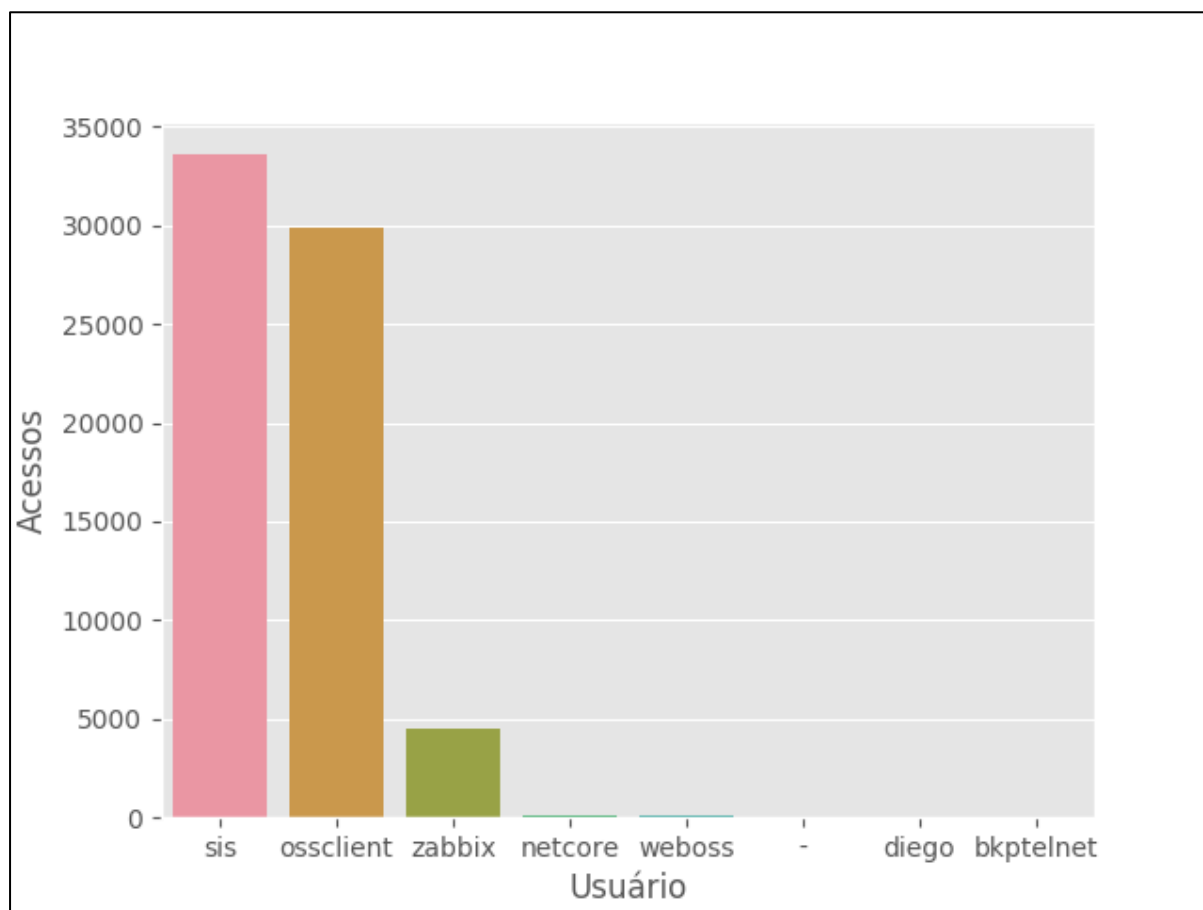
Como é possível visualizar na Figura 18, a variável “*acessos\_por\_url*” recebe os mesmos dados coletados que é possível observar na Figura 11 do capítulo 3.2. O módulo Matplotlib foi utilizado para fixar e salvar a imagem, na FIGURA 18 é possível ver este módulo simplificado como “plt”, Seaborn foi utilizado para transformar o dicionário em um gráfico, usando a função “*sns.barplot*” foi possível gerar um gráfico de barras e em seguida salvar usando a função “*savefig*” do Seaborn, desta forma é gerado uma imagem no formato PNG, essa imagem que contém o gráfico é salva em uma pasta chamada “static” que está contida dentro do projeto do Netstats, essa imagem será utilizada no código fonte HTML, desta forma é feito para todas as abstrações, tanto para os *logs* de acessos quanto para os *logs* de operações.

## 4. RESULTADO

O objetivo final deste projeto é transformar todos aqueles dados em alguma informação útil, como resultado os gráficos são gerados para uma fácil compreensão por qualquer pessoa que tenha o conhecimento básico de um *webservice* de provisionamento de redes.

Foram criados dois tipos de gráficos no Netstats, gráficos de barra e gráficos de pizza. Na Figura 19 é possível visualizar um exemplo de um dos gráficos de barra do Netstats.

Figura 19: Gráfico de barras



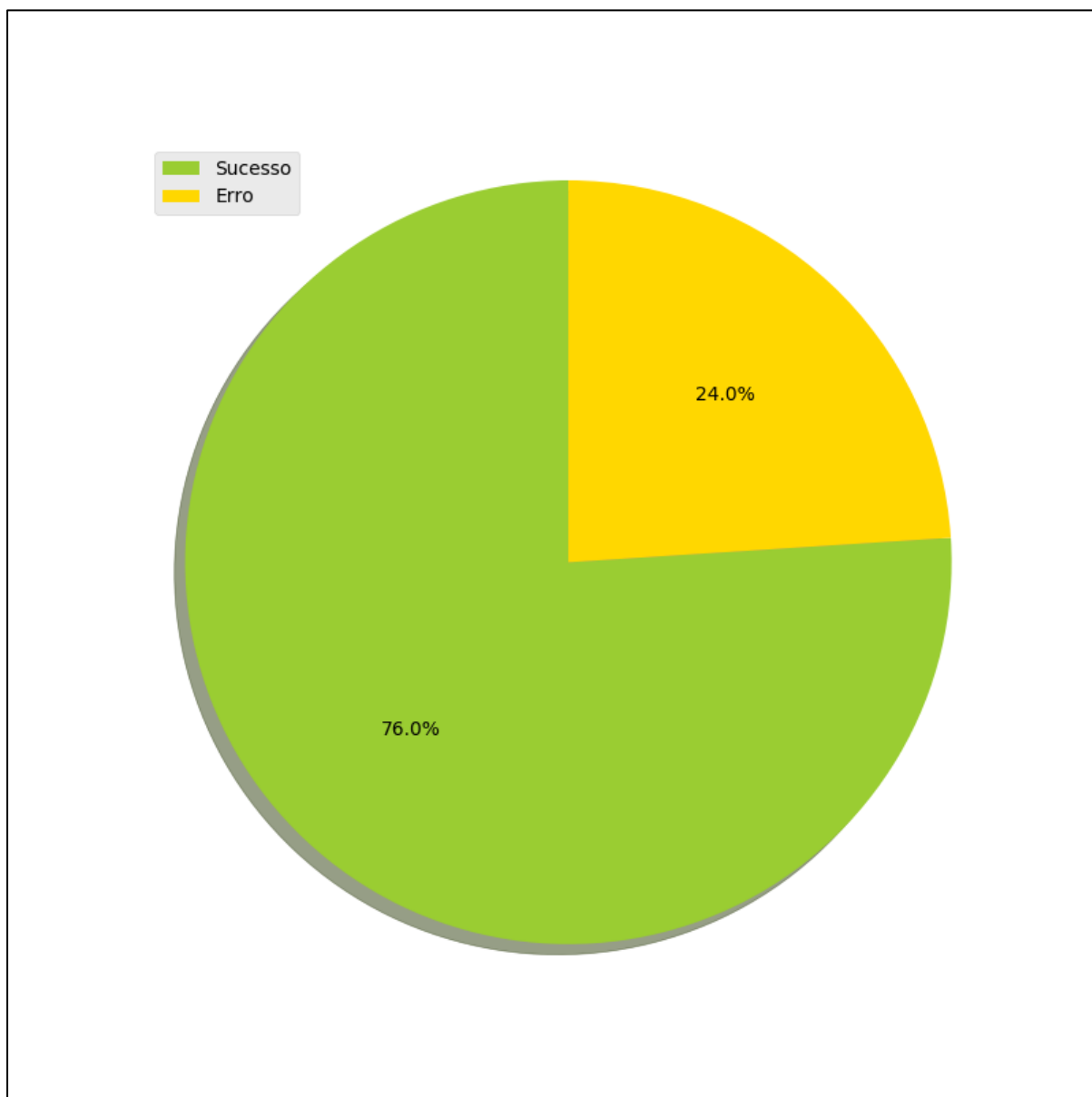
Fonte: Gabriel da Silva Lira (2019)

Observe que o gráfico da Figura 19 existem dois eixos, no eixo horizontal está o usuário, entenda usuário como a API que está utilizando o *webservice* conforme

descrito no capítulo 3.2, já no eixo vertical está a frequência de acesso para cada usuário.

Na Figura 20 é possível ver o exemplo de um dos gráficos de pizza gerados pelo Netstats.

Figura 20: Gráfico de pizza



Fonte: Gabriel da Silva Lira (2019)

É possível verificar através da Figura 20 que um gráfico de pizza é um gráfico em formato de círculo em que cada elemento consome uma parte, no caso do gráfico gerado na Figura 20, operações que obtiveram erro ou sucesso são comparados

dentro de um percentual, este gráfico é ideal para verificar de uma maneira fácil a disparidade entre as possíveis consequências de uma operação.

## 5. CONSIDERAÇÕES FINAIS

O objetivo geral deste projeto é demonstrar como é possível extrair informações de bases de dados independente do seu tamanho, e mesmo que possuam milhares de linhas ou que não estejam de forma sequencial e organizada.

A opção por trabalhar sob uma base de dados de um *webservice* de provisionamento de rede, um dos mais complexos tipos de estrutura de software, demonstra que é possível extrair diversas informações que com certeza serão muito relevantes para verificar o funcionamento da empresa, realizar tomada de decisões e até possíveis prospecções.

Ao longo do projeto foi possível verificar o poder da linguagem de programação Python, e não apenas isto, mas também foi possível observar a quantidade de bibliotecas que a comunidade *open source* desenvolve facilitando a vida acadêmica e profissional de diversas pessoas.

Este projeto não utilizou algoritmos específicos de ciência de dados, somente as ferramentas, o desenvolvimento do Netstats foi pensado em algo específico para sistemas de provisionamento de redes de computadores, apesar deste trabalho não apresentar longos estudos sobre os algoritmos já criados para ciência de dados, este deve ser considerado um projeto de ciência de dados, pois foi apresentado todos o processo de transformação de dados em conhecimento, a coleta, análise e visualização de dados, mesmo que tenha sido pensado de forma específica para um provedor de redes.

### 5.1 TRABALHOS FUTUROS

O repositório com o código fonte do projeto está disponível no link: "<https://github.com/liragabriel/netstats>" com total permissão para ser copiado e modificado.

Com algumas adaptações no código, o projeto pode ser utilizado em outros sistemas que não seja provisionamento de redes, com exceção das abstrações da classe "*Error*", que são compatíveis e fazem sentido apenas com APIs de provisionamento de redes de computadores.



## REFERÊNCIAS BIBLIOGRÁFICAS

AMARAL, Fernando. Introdução à ciência de dados: mineração de dados e big data. Rio de Janeiro: Alta Books, 2016.

CANALTECH. O que é API?, c2019. Disponível em: <<https://canaltech.com.br/software/o-que-e-api/>>. Acesso em: 16 de nov. de 2019.

ELKNER, j., DOWNEY, A., & MEYERS,. *Think Like a Computer Scientist: Learning with Python*. Samurai Media Limited, 2002.

MARTIN, j., Managing the Data Base Environment. Berkeley: Pearson Education, 1983.

MATPLOTLIB. Matplotlib: Python plotting, c2012. Página inicial. Disponível em: <<https://matplotlib.org/>>. Acesso em: 16 de nov. de 2019.

ORENSTEIN, David. Application Programming Interface. Disponível em: <<https://www.computerworld.com/article/2593623/application-programming-interface.html>> Acesso em: 23 de junho de 2019.

SAMUEL, S. (s.d.). Data visualization: A wise investment in your big data future. Disponível em<[https://www.sas.com/en\\_us/insights/articles/analytics/data-visualization-a-wise-investment-in-your-big-data-future.html](https://www.sas.com/en_us/insights/articles/analytics/data-visualization-a-wise-investment-in-your-big-data-future.html)>. Entrevista concedida a SAS (Statistical Analysis System).

SEABORB. Seaborn: Visualização de dados estatísticos, c2012. Página inicial. Disponível em: < [seaborn.pydata.org](https://seaborn.pydata.org)>. Acesso em: 16 de nov. de 2019.

## 6. APÊNDICES

### 6.1 APÊNDICE A – Código fonte do Netstats

app.py

```
import os
import pandas as pd
from flask import Flask, render_template, request
from netstats.access import Access
from netstats.fsan import Fsan
from netstats.lista_dataframe import ListaDataframe
from netstats.error import Error
from netstats.estatisticas_gerais import EstatisticasGerais

logs = pd.read_csv(r'static/websvc_access.csv')
logs.drop('fora', inplace=True, axis=1)
operacao = pd.read_csv(r'static/websvc_error1.csv')

class Netstats:

    def __init__(self):
        self.access = Access(logs)
        self.fsan = Fsan(operacao)
        self.lista_data = ListaDataframe(operacao)
        self.error = Error(operacao)
        self.estatisticas = EstatisticasGerais(operacao)

app = Flask(__name__)
netstats = Netstats()

@app.route('/')
@app.route('/home')
def home():
    for imagem in os.listdir('static'):
        if imagem in os.listdir('static'):
            if imagem != 'estilo.css' and imagem != 'fontAwesome' and imagem != 'websvc_access.csv' and imagem != 'websvc_error1.csv':
                os.remove(f'static/{imagem}')

    resposta = netstats.estatisticas.estatisticas()
```

```

    return render_template('home.html', data=resposta)

@app.route('/pesquisar-fsan', methods=['POST', 'GET'])
def pesquisar_fsan():

    if request.method == 'POST':

        fsan_value = request.form['fsan']

        for tabela in netstats.lista_data.dataframe():
            if fsan_value == tabela.columns:
                resposta = tabela.to_json(orient='values')
                break
            else:
                resposta = 'FSAN não identificado'
        return render_template('pesquisar_fsan.html', resposta=resposta, fsan=
fsan_value)

    else:
        return render_template('pesquisar_fsan.html')

@app.route('/analises')
def analises():

    while True:
        if not os.path.exists('static/acessos_por_usuario.png'):
            netstats.access.graph_acesso_por_usuario()

        if not os.path.exists('static/acessos_por_url.png'):
            netstats.access.graph_acesso_por_url()

        if not os.path.exists('static/status_code.png'):
            netstats.access.graph_status_code()

        if not os.path.exists('static/percentual_sucesso.png'):
            netstats.error.percentual_sucesso()

        if not os.path.exists('static/operacao_sucesso.png'):
            netstats.error.sucesso_por_operacao()

        if not os.path.exists('static/operacao_error.png'):
            netstats.error.erros_por_operacao()

    else:
        return render_template('analises.html')

```

```

@app.route('/acessos-por-usuario')
def acesso_por_usuario():
    return render_template('acessos_por_usuario.html',
                           data=netstats.access.data_acesso_por_usuario())

@app.route('/acessos-por-url')
def rota_acesso_por_url():
    return render_template('acessos_por_url.html',
                           data=netstats.access.data_acesso_por_url())

@app.route('/status-code')
def rota_status_code():
    return render_template('status_code.html', data=netstats.access.data_status_code())

@app.route('/percentual-sucesso')
def rota_percentual_sucesso():
    return render_template('percentual_sucesso.html')

@app.route('/sucesso-por-operacao')
def rota_sucesso_por_operacao():
    return render_template('sucesso_por_operacao.html')

@app.route('/erros-por-operacao')
def rota_erros_por_operacao():
    return render_template('operacao_error.html')

if __name__ == '__main__':
    app.run(debug=True)

```

## access.py

```

import seaborn as sns
from matplotlib import pyplot as plt
plt.style.use('ggplot')

class Access:

    def __init__(self, logs):
        self.logs = logs

```

```

def graph_acesso_por_usuario(self):

    """
        Cria um arquivo png com o gráfico representando a quantidade de ac
        essos por API.

        Returns
        -----
            None
    """

    acessos_por_usuario = self.logs.usuario.value_counts().to_frame().rese
t_index()
    acessos_por_usuario.columns = ['Usuário', 'Acessos']

    plt.figure()
    plt.tight_layout()
    graf_cod = sns.barplot(x=acessos_por_usuario['Usuário'], y=acessos_por
_usuario['Acessos'])
    graph = graf_cod.get_figure()
    graph.savefig('static/acessos_por_usuario.png')

def data_acesso_por_usuario(self):

    """
        Cria um dataframe com a quantidade de acessos por API.

        Returns
        -----
            dataframe
    """

    acessos_por_usuario = self.logs.usuario.value_counts().to_frame().rese
t_index()
    acessos_por_usuario.columns = ['Usuário', 'Acessos']
    data = acessos_por_usuario.head().to_html()

    return data

def graph_acesso_por_url(self):

    """
        Cria um arquivo png com o gráfico representando a quantidade de ac
        essos por URL.

        Returns
        -----
    """

```

```

        """
        None

        """

        urls = self.logs.url.value_counts().to_frame().reset_index()
        urls.columns = ['URL', 'Acessos']

        x = urls.loc[(urls['Acessos'] >= 4000)].URL
        y = urls.Acessos

        plt.figure()
        plt.xticks(rotation=45)
        graf_url = sns.barplot(x=x, y=y)
        plt.tight_layout()
        fig = graf_url.get_figure()
        fig.savefig('static/acessos_por_url.png', dpi=300, bbox_inches='tight'
    )

def data_acesso_por_url(self):
    """
        Cria um dataframe com a quantidade de acessos por URL.

        Returns
        -----
        dataframe
    """

    urls = self.logs.url.value_counts().to_frame().reset_index()
    urls.columns = ['URL', 'Acessos']
    data = urls.head().to_html()

    return data

def graph_status_code(self):
    """
        Cria um arquivo png com o gráfico representando a frequência de ca
da status code.

        Returns
        -----
        None
    """

    status = self.logs.status_code.value_counts().to_frame().reset_index()
    status.columns = ['Code', 'Frequência']

```

```

        status.Code = [int(i) for i in status.Code]

        plt.figure()
        plt.tight_layout()
        graf_cod = sns.barplot(x=status['Code'].astype(int), y=status['Frequên
cia'])
        fig = graf_cod.get_figure()
        fig.savefig('static/status_code.png')

    def data_status_code(self):
        """
        Cria um dataframe com a frequência de cada status code.

        Returns
        -----
        dataframe
        """

        status = self.logs.status_code.value_counts().to_frame().reset_index()
        status.columns = ['Code', 'Frequência']

        status.Code = [int(i) for i in status.Code]
        data = status.head().to_html()

        return data

```

## error.py

```

import pandas as pd
from matplotlib import pyplot as plt
from netstats.lista_mensagens import ListaMensagens

class Error:

    def __init__(self, operacao):
        self.operacao = operacao

    def percentual_sucesso(self):
        """
        Cria um arquivo png com o gráfico representando o percentual de su
cessos por operação.

```

```

        Returns
        -----
        None
    """

    ultima_msg = ListaMensagens(self.operacao).mensagem()

    contador_creates = 0
    for item in ultima_msg:
        if 'OK' in item or 'onu_business_create' in item or 'voip_create:'
in item or '\
        onu_delete: fsan' in item:
            contador_creates += 1

    cruzo_creates = contador_creates*100
    resultado_sucesso = cruzo_creates/len(ultima_msg)

    #Percentual de ERROR
    contador_error = 0
    for item in ultima_msg:
        if 'error' in item:
            contador_error += 1

    cruzo_error = contador_error*100
    resultado_error = cruzo_error/len(ultima_msg)

    ocorrencia = {
        'tipo': ['Sucesso', 'Erro'],
        'quantidade': [resultado_sucesso, resultado_error]
    }

    data_ocorrencia = {
        'Resultado': ['Sucesso', 'Erro'],
        'FSANs': [contador_creates, contador_error]
    }

    data_ocorrencia = pd.DataFrame(data_ocorrencia)

    labels = ocorrencia['tipo']
    sizes = ocorrencia['quantidade']
    colors = ['yellowgreen', 'gold']
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.pie(sizes, colors=colors, shadow=True,
           startangle=90, autopct='%1.1f%%')
    ax.legend(labels, loc="best")
    ax.axis('equal')
    plt.savefig('static/percentual_sucesso.png')

```



```

def sucesso_por_operacao(self):
    """
        Cria um arquivo png com o gráfico representando a quantidade de op
        erações bem-sucedidas

        para cada operação.

        Returns
        -----
        None
    """

    ultima_msg = ListaMensagens(self.operacao).mensagem()

    sucessos = []
    for i in range(len(ultima_msg)):
        if 'error' not in ultima_msg[i]:
            corte_sucessos = ultima_msg[i].split()
            sucessos.append(corte_sucessos[0])

    lista_sucessos = []
    for item in sucessos:
        if item not in lista_sucessos:
            lista_sucessos.append(item)

    dic = {
        'smart': 0,
        'onu_delete': 0,
        'onu_business_create': 0,
        'voip_create': 0
    }

    for item in sucessos:
        for operacao in dic:
            if item == operacao:
                dic[operacao] += 1

    quantidade_sucessos = {
        'Função': [
            'onu_home_create', 'onu_delete', 'voip_create', 'onu_business_
create'
        ],
        'Quantidade': [
            dic['smart'], dic['onu_delete'], dic['voip_create'], dic['on
u_business_create']
        ]
    }

```

```

quantidade_sucessos = pd.DataFrame(quantidade_sucessos)

labels = quantidade_sucessos['Função']
sizes = quantidade_sucessos['Quantidade']
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(sizes, colors=colors, shadow=True)
ax.legend(labels, loc="best")
ax.axis('equal')
plt.savefig('static/operacao_sucesso.png')

def erros_por_operacao(self):
    """
        Cria um arquivo png com o gráfico representando a quantidade de op
        erações malsucedidas

        para cada operação.

        Returns
        -----
        None
    """

    ultima_msg = ListaMensagens(self.operacao).mensagem()

    erros = []
    for i in range(len(ultima_msg)):
        if 'error' in ultima_msg[i]:
            corte_erros = ultima_msg[i].split()
            erros.append(corte_erros[0])

    lista_erros = []
    for item in erros:
        if item not in lista_erros:
            lista_erros.append(item)

    dic = {
        'DELETE': 0,
        'onu_bridge_path_list': 0,
        'onu_resync_update': 0,
        'omci_onu_status': 0,
        'CREATE': 0,
        'wifi_update': 0,
        'onu_status': 0,
        'onu_set2default_update': 0,
        'onu_checa_status': 0,
    }

```

```

        'dslam_fsan_status': 0,
        'onu_check_conf_status': 0,
    }

    for item in erros:
        for operacao in dic:
            if item == operacao:
                dic[operacao] += 1

    quantidade_erros = {
        'Função': [
            'DELETE', 'onu_bridge_path_list', 'onu_check_conf_status', 'onu_checa_status',
            'omci_onu_status', 'CREATE', 'onu_resync_update', 'onu_set2default_update',
            'dslam_fsan_status', 'wifi_update', 'onu_status'
        ],
        'Quantidade': [
            dic['DELETE'], dic['onu_bridge_path_list'], dic['onu_resync_update'],
            dic['omci_onu_status'], dic['CREATE'], dic['wifi_update'],
            dic['onu_status'], dic['onu_set2default_update'],
            dic['onu_checa_status'], dic['dslam_fsan_status'],
            dic['onu_check_conf_status']
        ]
    }

    quantidade_erros = pd.DataFrame(quantidade_erros)

    labels = quantidade_erros['Função']
    sizes = quantidade_erros['Quantidade']
    colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'crimson', 'darkblue',
              'fuchsia', 'sienna', 'tan', 'orangered', 'dimgray']
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.pie(sizes, colors=colors, shadow=True)
    ax.legend(labels, loc="best")
    ax.axis('equal')
    plt.savefig('static/operacao_error.png')

```

fsan.py

```

class Fsan:

    def __init__(self, operacao):
        self.operacao = operacao

```

```

def lista_de_fsans(self):
    """
        Retorna uma lista com todos FSANs na base de dados.

        Returns
        -----
        list
    """

    lista_com_todos = []
    for i in range(self.operacao.index.max()):
        lista_com_todos.append(self.operacao.operacao.loc[(self.operacao.o
peracao.index ==
                                                                    i)].str.split()
)

    lista_com_fsan = []
    for i in range(self.operacao.index.max()):
        if 'fsan' in lista_com_todos[i][i]:
            lista_com_fsan.append(lista_com_todos[i][i])

    lista_fsan = []
    for i in range(len(lista_com_fsan)):
        for j in range(len(lista_com_fsan[i])):
            if 'fsan' in lista_com_fsan[i][j] and 'dslam_fsan_status:' not
in lista_com_fsan[i][j]:
                lista_fsan.append(lista_com_fsan[i][j+1])

    fsan = []
    for item in lista_fsan:
        if item.endswith(':'):
            item = item[:-1]
        if item not in fsan:
            fsan.append(item)

    return fsan

```

### lista\_dataframe.py

```

import pandas as pd
from netstats.fsan import Fsan

class ListaDataframe:

    def __init__(self, operacao):

```

```

        self.operacao = operacao

    def dataframe(self):
        """
        Retorna uma lista de dataframes por FSAN, cada dataframe contém as
        operações realizadas

        com a FSAN.

        Returns
        -----
        list
        """

        fsan = Fsan(self.operacao).lista_de_fsans()

        sequencia = []
        for i in fsan:
            lista = []
            for j in self.operacao.operacao:
                if i in j or i+':' in j:
                    lista.append(j)
            sequencia.append(lista)

        lista_data = []
        for i in sequencia:
            lista_data.append(pd.DataFrame(i))
            pd.set_option('display.max_colwidth', -1)

        for i in range(len(lista_data)):
            lista_data[i].columns = [fsan[i]]

        return lista_data

```

### lista\_mensagens.py

```

from netstats.fsan import Fsan
from netstats.lista_dataframe import ListaDataframe

class ListaMensagens:

    def __init__(self, operacao):
        self.operacao = operacao

```

```

def mensagem(self):
    """
        Retorna lista com a última mensagem de cada dataframe de FSANs.

        Returns
        -----
        list
    """

    fsan = Fsan(self.operacao).lista_de_fsans()
    dataframe = ListaDataframe(self.operacao).dataframe()

    ultima_msg = []
    for i in range(len(dataframe)):
        ultima_msg.append(dataframe[i].loc[dataframe[i].index.max(), fsan[
i]])

    return ultima_msg

```

estatisticas\_gerais.py

```

from netstats.lista_dataframe import ListaDataframe

class EstatisticasGerais:

    def __init__(self, operacao):
        self.operacao = operacao

    def estatisticas(self):

        lista_data = ListaDataframe(self.operacao).dataframe()

        lista_com_todos = []
        for x in range(self.operacao.index.max()):
            lista_com_todos.append(self.operacao.operacao.loc[(self.operacao.o
peracao.index == x)].str.split())

        lista_com_fsan = []
        for x in range(self.operacao.index.max()):
            if 'fsan' in lista_com_todos[x][x]:
                lista_com_fsan.append(lista_com_todos[x][x])

        lista_fsan = []
        for c in range(len(lista_com_fsan)):
            for x in range(len(lista_com_fsan[c])):

```

```

        if 'fsan' in lista_com_fsan[c][x] and 'dslam_fsan_status:' not
in lista_com_fsan[c][x]:
            lista_fsan.append(lista_com_fsan[c][x+1])

fsan = []
for item in lista_fsan:
    if item.endswith(':'):
        item = item[:-1]
    if item not in fsan:
        fsan.append(item)

operacoes = len(lista_com_fsan)
fsans = len(fsan)
media = len(lista_com_fsan) / len(fsan)

maior = []
for item in lista_data:
    maior.append(item.index.max())
res_maior = max(maior)

data_stats_success = {
    'Quantidade de operações': int(operacoes),
    'Quantidade de fsans': int(fsans),
    'Média de operações por fsan': int(media),
    'Maior número de operações em uma fsan': int(res_maior),
}

return data_stats_success

```

## 6.2 APÊNDICE B – Licença de software

### MIT License

Copyright (c) 2019 Gabriel Lira

Permission is hereby granted, free of charge, to **any** person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above **copyright** notice and this permission notice shall be included in **all** copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.