UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOHNY DE SOUZA RAMOS
LUCAS GUILHERME DE SOUZA BORGES NASCIMENTO
ROBERTO ANDREI BISCHOFF

FERRAMENTA PARA MONITORAMENTO DE DISPOSITIVOS DE REDE IMPLEMENTADA UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON

CURITIBA
2022

JOHNY DE SOUZA RAMOS
LUCAS GUILHERME DE SOUZA BORGES NASCIMENTO
ROBERTO ANDREI BISCHOFF

FERRAMENTA PARA MONITORAMENTO DE DISPOSITIVOS DE REDE IMPLEMENTADA UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON

Tool for monitoring network devices implemented using the python programming language

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Tecnólogo em Sistemas de Telecomunicações do Curso Superior de Tecnologia em Sistemas de Telecomunicações da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. M. Sc. Omero Francisco Bertol

CURITIBA
2022

JOHNY DE SOUZA RAMOS
LUCAS GUILHERME DE SOUZA BORGES NASCIMENTO
ROBERTO ANDREI BISCHOFF

FERRAMENTA PARA MONITORAMENTO DE DISPOSITIVOS DE REDE IMPLEMENTADA UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Tecnólogo em Sistemas de Telecomunicações do Curso Superior de Tecnologia em Sistemas de Telecomunicações da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 14 de Junho 2022

Omero Francisco Bertol
Mestrado
Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Curitiba

---

Edenilson José da Silva
Doutorado
Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Curitiba

---

Kleber Kendy Horikawa Nabas
Doutorado
Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Curitiba

CURITIBA

2022

RESUMO

As redes de computadores estão cada vez mais presentes em todas as organizações, sejam públicas ou privadas. Juntamente com esse aumento se faz necessário, cada vez mais, a utilização de ferramentas para controle e monitoramento dessas redes. Somente assim é possível identificar falhas de funcionamento e de segurança antes que estes acarretem prejuízos para a empresa. O objetivo deste trabalho é trazer uma abordagem prática para o desenvolvimento de uma ferramenta simples de monitoramento de dispositivos de rede fazendo o uso do protocolo *Simple Network Management Protocol* (SNMP) para a comunicação e da linguagem de programação Python que foi escolhida para ser utilizada nesse projeto devido a sua ampla utilização e conteúdo disponibilizado pela comunidade.

Palavras-chave: Redes de computadores; Monitoramento; SNMP; Python; Protocolos.

ABSTRACT

Computer networks are increasingly present in all organizations, whether public or private. Along with this increase, it is increasingly necessary to use tools to control and monitor these networks. Only in this way is it possible to identify operational and safety failures before they cause damage to the company. The objective of this work is to bring a practical approach to the development of a basically network monitoring tool, making use mainly of the SNMP protocol for communication and the Python programming language that was chosen to be used in this project due to its wide use and content made available by the community.

Keywords: Computer network; Monitoring; SNMP; Python; Protocols.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface* (ou Interface de Programação de Aplicação)
CLI	*Command-Line Interface* (ou Interface de Linha de Comando)
CNRI	*Corporation for National Research Initiatives*
CSS	*Cascading Style Sheets*
CSRF	*Cross Site Request Forgery*
CVS	*Concurrent Version System*
HTML	*Hiper Text Markup Language* (ou Linguagem de Marcação de Hipertexto)
IDE	*Integrated Development Environment* (ou Ambiente de Desenvolvimento Integrado
IP	*Internet Protocol*
LDAP	*Lightweight Directory Access Protocol*
MIB	*Management Information Bases*
OID	*Object IDentifier*
PEP	*Python Enhancement Proposal*
PSF	*Python Software Foundation*
RPC	*Remote Procedure Call*
SIG	*Special Interest Group*
SNMP	*Simple Network Management Protocol*
SPA	*Single Page Application*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol* / *Internet Protocol*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locators*
VS Code	*Visual Studio Code*
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Falbriard (2001), os protocolos utilizados em redes de comunicação definem os conjuntos de regras que coordenam e asseguram o transporte das informações úteis entre dois ou mais dispositivos conectados. Visando o crescimento exponencial da utilização dos diversos tipos de protocolos de rede para transmissão de dados, torna-se primordial que tais protocolos tenham níveis cada vez mais altos de segurança, desempenho e disponibilidade.

Neste estudo serão abordados os protocolos *Transmission Control Protocol* (TCP), *User Datagram Protocol* (UDP) e o *Simple Network Management Protocol* (SNMP).

O protocolo TCP é um padrão de comunicação confiável e muito utilizado sendo orientado à conexão, e isso significa que antes de começar a enviar efetivamente a informação, ele estabelece uma conexão para garantir que toda a informação enviada esteja de fato sendo recebida do outro lado (MONTE; BERNARDO; OLIVEIRA, 2005).

Já o UDP é usado para aplicações que necessitam de transporte rápido pois não faz verificações de entrega, ordenação e controle dos pacotes, porém não garante que o receptor da comunicação recebeu a informação; essa solução pode parecer pouco ortodoxa, mas é muito utilizada justamente pela simplicidade e pouca geração de tráfego na rede (MONTE; BERNARDO; OLIVEIRA, 2005).

Quando se fala do protocolo SNMP, as versões 1 e 2 possuem falhas graves pois não contam com ferramentas de encriptação em sua comunicação, sendo assim é possível interceptar os pacotes e identificar informações destes como, por exemplo, a *community* e comandos de *get* e *set*. Apesar de a versão 3 do protocolo SNMP já ter corrigido essa falha, ela ainda é muito pouco utilizada por conta da grande popularidade das versões anteriores que ficaram enraizadas nas redes (ALVARENGA; RAMOS, 2011).

Neste contexto, este projeto tem por objetivo desenvolver uma ferramenta, utilizando a linguagem de programação Python, para análise de desempenho e identificação de falhas em aplicações e hardware utilizando o protocolo SNMP.

1.1 Contextualização

O desenvolvimento de uma ferramenta de monitoramento de rede utilizando alguns dos principais protocolos de comunicação será realizado utilizando uma das linguagens de programação mais populares atualmente, o Python.

Os protocolos a serem utilizados pela ferramenta foram escolhidos devido, principalmente, a sua simplicidade e robustez. O SNMP, por exemplo, é um protocolo extremamente flexível, abrindo uma gama de possibilidades de implementação de funcionalidades para a ferramenta a ser desenvolvida.

Já a linguagem de programação Python possui uma infinidade de bibliotecas e muito material de consulta, principalmente em fóruns da comunidade na *internet*. Isso faz com que seja uma linguagem com muitas alternativas para solução de problemas que surgirão durante o desenvolvimento.

1.2 Problema

A utilização cada vez maior de tráfego de rede por empresas privadas e por instituições do setor público faz com seja, cada vez mais, imprescindível que se tenha o total controle desse imenso tráfego.

Quanto maior o tráfego de rede, maior a possibilidade de falhas ou ataques à rede. A indisponibilidade geralmente causa prejuízos consideráveis tanto de valor monetário como de imagem para as empresas e instituições.

Este trabalho visa o desenvolvimento de uma ferramenta para análise de rede de forma personalizada e utilizando protocolos de comunicação simples e desenvolvido de forma prática. Com a sua utilização, o gestor de tecnologia da informação poderá ter um panorama em tempo real de tudo o que acontece na rede sob sua responsabilidade.

1.3 Objetivos

Nesta seção são apresentados os objetivos gerais e específicos deste Trabalho de Conclusão de Curso (TCC) de graduação.

1.3.1 Objetivo geral

Desenvolver uma ferramenta, utilizando a linguagem de programação Python, para análise de desempenho e identificação de falhas em dispositivos componentes de uma rede de computadores.

1.3.2 Objetivos específicos

As etapas a serem realizadas para atingir-se o Objetivo Geral deste TCC são:

- Escolher e estudar sobre os principais protocolos de comunicação de rede;
- Estudar sobre a linguagem de programação Python, suas bibliotecas e disponibilidade de conteúdo;
- Estudar sobre ferramentas adicionais para comunicação, gerenciamento de filas e banco de dados;
- Desenvolver o *back-end* da ferramenta utilizando Python;
- Fazer a comunicação entre a ferramenta e o banco de dados;
- Desenvolver o *front-end* da ferramenta;
- Testar e analisar o desempenho da ferramenta finalizada.

1.4 Justificativa

Este estudo e desenvolvimento se justifica por se tratar de um tema extremamente importante atualmente: As redes de computadores e a importância de se ter o controle sobre o que se trafega nela.

Em grandes empresas o tráfego é altíssimo e a probabilidade de falhas é igualmente alto.

Aplicações para controle dessas redes tornaram-se ferramentas de trabalho indispensáveis para as equipes de tecnologia da informação, pois auxiliam na identificação de falhas, controle de fluxo e em decisões sobre melhoria de equipamentos de rede.

O desenvolvimento de uma ferramenta personalizada permite que sejam incorporadas funções a ela, atendendo assim a demandas específicas de cada empresa.

1.5 Estrutura do trabalho

Esta monografia de especialização está dividida em 7 (sete) seções. Nesta primeira seção foi introduzido o assunto tema do trabalho e também foram abordados a motivação e os objetivos geral e específicos da pesquisa, a justificativa e a estrutura geral do trabalho.

Já na segunda seção é feita uma revisão de literatura, com uma análise do que já foi escrito sobre as ferramentas e protocolos a serem utilizados para o desenvolvimento da ferramenta de rede.

Na terceira seção se tem a descrição de todas a ferramentas e linguagens de programação que serão utilizadas, trazendo um panorama histórico e suas principais funcionalidades. Nessa seção há a explicação sobre: Python, Typescript, Pycharm, Visual Studio Code, Django, Angular, Celery, Rabbit MQ, HTML e CSS.

A quarta seção traz o desenvolvimento da ferramenta em si, com detalhes referentes ao *back-end*, banco de dados e *front-end.* Nela podemos observar grande parte do código-fonte e a explicação das tabelas de banco de dados utilizadas, bem como as telas finais da aplicação.

As considerações são detalhadas na quinta e última seção e traz um panorama dos desafios e aprendizados que se teve durante o desenvolvimento do trabalho.

2 REVISÃO DA LITERATURA

Segundo Tanenbaum (2011, p. 456), o sistema de transferência de mensagens tem como objetivo transmitir mensagens do remetente ao destinatário. Destacando também, que a maneira mais simples de fazer isso é estabelecer uma conexão de transporte entre a máquina de origem e a de destino e, em seguida, transferir a mensagem. Após a análise de como isso é feito normalmente, será realizada um exame de algumas situações nas quais essa opção não funciona, mostrando o que pode ser feito para contornar o problema.

As redes de computadores, segundo McKeown *et al.* (2008), se tornaram parte integrante da sociedade atual. Destaca-se também, que o número de aplicações online é cada vez maior e os projetos de redes devem suportar essa demanda, mesmo que não tenha sido previsto tamanho crescimento. Para que isso seja possível, muitas vezes, são adotadas soluções provisórias que acabam gerando problemas posteriormente, pois os softwares dos equipamentos ativos de rede continuam os mesmos pelo fato dos aparelhos não suportarem uma atualização por limitações de hardware. Isso traz um efeito chamado de “ossificação” da rede.

As redes atuais seguem o modelo TCP/IP (*Transmission Control Protocol* / *Internet Protocol*) que apresenta um conjunto de protocolos para comunicação entre dispositivos e estes são organizados em uma espécie de pilha de camadas na qual as camadas inferiores oferecem serviços às superiores. Esse processo facilita o entendimento para implementação de novas tecnologias e a interconectividade entre sistemas (KLEIS, 2019).

O SNMP pode ser implementado facilmente em uma rede, pois consome poucos recursos de hardware. Ele serve, principalmente, para trocar informações de gerenciamento entre os dispositivos conectados, facilitando assim a identificação e correção rápida de anomalias na rede (CARYULY, 2004).

3 FERRAMENTAS UTILIZADAS

Nas subseções a seguir veremos as ferramentas que foram utilizadas para o desenvolvimento do sistema de monitoramento de rede. Além disso há informações referentes a cada uma delas.

3.1 Linguagem de programação python

A linguagem de programação Python é, atualmente, a terceira linguagem de programação mais utilizada no mundo e, apesar de ser muito simples, para se ter domínio é necessário muito treino e estudo.

Foi criada no final dos anos 80 e o responsável foi o holandês Guido Van Rossum. Além de criador, Guido foi o responsável pela definição de todas as formas possíveis de evolução da linguagem.

A linguagem Python é fortemente inspirada na linguagem ABC que surgiu no início dos anos 80 como uma alternativa para a linguagem BASIC (TOKIO, 2021).

Na Figura 1, tem-se um exemplo que demonstra a simplicidade da linguagem ABC.

Figura 1 – Exemplo de código na linguagem ABC

```
HOW TO RETURN words document:
        PUT {} IN collection
        FOR line IN document:
                FOR word IN split line:
                        IF word not.in collection
                                INSERT word IN collection
        RETURN collection
```

Fonte: Tokio (2021).

3.1.1 Primeira release da linguagem de programação python

A primeira versão pública da linguagem Python foi lançada em fevereiro de 1991. Versão essa de número 0.9.0.

Já em sua primeira versão, Python já possuía funcionamento modular, o que permitia que a linguagem fosse muito mais limpa e acessível. Além disso, já se tinha classes com heranças, tratamento de exceções e funções (TOKIO, 2021).

3.1.2 Origem do nome python

Diferentemente do que muitos imaginam, o nome da linguagem não surgiu como referência à cobra píton e sim seguindo a uma tradição de utilizar nomes de programas de TV da época. Trata-se do Monty Python's Flying Circus. Porém, pouco tempo depois, Guido acabou acatando a sugestão da editora O'Reilly de utilizar o animal na capa de seu primeiro livro sobre a linguagem (TOKIO, 2021).

3.1.3 Evolução da linguagem

Guido Van Rossum começou o desenvolvimento da linguagem Python enquanto trabalhava em um centro de pesquisa chamado CWI, sendo que esse centro lançou, em 1995, a versão 1.2 da linguagem.

Já no ano 2000, a equipe principal de desenvolvimento do Python se desvinculou do CWI e migrou para a BeOpen.com, formando a equipe BeOpen Python Labs.

A versão 2.0 foi lançada no ano 2000 e nela foi inserida uma das características mais importantes do Python: A geração de listas. Ainda nessa versão foi implementado a capacidade de fazer referências cíclicas, assim o lixo gerado pelo código pode ser recolhido automaticamente.

No ano de 2008 ocorreu a última grande atualização com o lançamento da versão 3.0 que corrigiu falhas de *design* e inseriu formas redundantes de programação, para que o mesmo elemento pudesse ser programado de diversas formas diferentes com o mesmo resultado final (TOKIO, 2021).

No Quadro 1, pode-se observar um comparativo com as versões do Python desde sua criação até o ano de 2000.

Quadro 1 – Versões da linguagem de programação Python

(continua)

MÊS/ANO	VERSÃO
fev/91	0.9.0
fev/91	0.9.1
out/91	0.9.2
dez/91	0.9.4
jan/92	0.9.5
abr/92	0.9.6
jan/93	0.9.8
jul/93	0.9.9
jan/94	1.0.0

Quadro 1 – Versões da linguagem de programação Python

(conclusão)

MÊS/ANO	VERSÃO
fev/94	1.0.2
mai/94	1.0.3
jul/94	1.0.4
out/94	1.1.0
nov/94	1.1.1
abr/95	1.2.0
out/95	1.3.0
out/96	1.4.0
jan/98	1.5.0
out/98	1.5.1
abr/99	1.5.2
set/00	1.6.0

Fonte: Autoria própria (2022).

A evolução da linguagem continuou e várias versões foram lançadas desde então sendo que hoje é uma das principais linguagens de programação mais utilizadas no mundo e, atualmente, está na versão 3.9.4 (TOKIO, 2021).

3.2 Linguagem de programação typescript

O TypeScript é uma linguagem de programação desenvolvida pela Microsoft. É um *superset* da linguagem Javascript e adiciona elementos que não são encontrados em Javascript como tipagem estática.

A linguagem TypeScript nasceu de um desejo interno da Microsoft. Equipes internas da empresa estavam desenvolvendo aplicações complexas em Javascript e desejam ter os mesmos tipos de ferramentas que os desenvolvedores de linguagens como C# estão acostumados - a capacidade de detectar erros em tempo de desenvolvimento, renomear métodos e variáveis e corrigir todos os lugares que esses métodos e variáveis são usados. Lançada originalmente em 2012 a linguagem continua sendo desenvolvida pela Microsoft e pela comunidade (ARS_TECHNICA, 2021).

3.2.1 Releases da linguagem de programação typescript

A primeira versão da Linguagem de Programação TypeScript liberada ao público foi a 0.8 (alpha) em outubro de 2012, após dois anos de desenvolvimento interno (JAVATPOINT, 2019).

No decorrer dos anos, diversas versões foram lançadas e a cada nova versão, foram implementadas melhorias de desempenho e usabilidade. Atualmente, o TypeScript está na versão 4.5 e a versão 4.6 já está em fase beta (JAVATPOINT, 2019).

3.2.2 Recursos

TypeScript possui vários recursos provenientes do ECMAScript 6 e alguns que não são encontradas em Javascript. Alguns desses recursos são:

- anotações de tipo e verificação de tipo em tempo de compilação;
- inferência de tipo;
- apagamento de tipo;
- *interfaces*;
- tipos enumerados;
- genéricos;
- *namespaces*;
- tuplas;
- assíncrono/aguardar.

3.2.3 Evolução da comunidade

Desde sua primeira versão a linguagem vem sendo continuamente desenvolvida tanto pelo time interno na Microsoft como pela comunidade em geral. Por ser uma linguagem de fonte aberta, qualquer um pode colaborar através do repositório que se encontra no GitHub. Apesar do lançamento ter ocorrido em 2012, apenas em 2014 começando com a versão 1.1 que o código fonte foi aberto para comunidade. Desde então já foram realizados 4 lançamentos grandes e a linguagem já se encontra em sua versão 4.x.x. A primeira após a versão 1.x.x foi a 2.x.x lançada em setembro de 2016, seguida pela versão 3.x.x lançada em julho de 2018 e em agosto de 2020 foi lançada a versão mais atual (GITHUB, 2019).

A versão beta do TypeScript 2 foi lançada em 11 de julho de 2016 e já apresentava várias novidades. Entre as novidades estavam Tipos não anuláveis, Análise de fluxo de controle para tipos, Declarações de módulo mais fáceis. Apenas em 22 de setembro do mesmo ano que a versão 2 foi oficialmente lançada. Além

dos recursos já mencionados o lançamento oficial também trouxe o Modificador *readonly* e corrigiu vários bugs encontrados durante seu tempo em Beta (ROSENWASSER, 2016).

Diferente da versão 2 a versão 3 do TypeScript não teve um período de beta sendo esse realizado em lançamentos posteriores da versão 2. Segundo Rosenwasser (2018), o lançamento em 30 de junho de 2018 da versão 3 contou com as seguintes melhorias:

- modo de construção;
- extração e disseminação de listas de parâmetros com tuplas;
- tipos de tupla mais rica e o tipo *unknown*;
- *unknown* é um nome de tipo reservado;
- erros e UX aprimorados;
- mensagens e elaboração aprimoradas;
- suporte para *defaultProps* em JSX;
- refatorações de importação nomeadas;
- correções rápidas para código inacessível e rótulos não utilizados;
- quebrando mudanças.

Lançada em 20 de agosto de 2020 e seguindo o mesmo padrão da versão 3, a versão 4 teve seu beta em versões subsequentes da versão 3 e sua primeira versão já foi a de lançamento. Apesar de não conter tantas novidades como a versão 3 essa última versão mesmo assim conta com a seguintes adições:

- Tipos de tupla variável e elementos de tupla rotulados;
- Classe de inferência de propriedade de construtores;
- Operadores de atribuição de curto-circuito;
- *unknown* nas cláusulas de captura;
- Fábricas JSX Customizadas;
- Melhorias de velocidade no modo de compilação com –noEmitOnError;
- --incremental com –noEmit.

O trabalho de desenvolvimento na versão 4 continua e sua versão mais recente é a 4.5 que foi lançada em 17 de novembro de 2021. A versão 5 deve ser lançada no primeiro semestre de 2022 e continuar com as melhorias a linguagem (ROSENWASSER, 2018).

3.3 Pycharm

O PyCharm é um Ambiente de Desenvolvimento Integrado (ou *Integrated Development Environment* - IDE) que fornece recursos como:

- complementação de código inteligente;
- inspeções de código;
- realce dinâmico de erros e correções rápidas;
- refatorações de código automatizada;
- recursos de navegação avançados.

Os recursos de auxílio ao desenvolvimento presentes no PyCharm são os mais diversos sendo que é possível a personalização da interface do usuário na ferramenta que variam desde as cores até as teclas de atalho.

Além disso, muitos plugins estão disponíveis (ao todo são mais de 50) sendo possível a integração com outras ferramentas de *framework*, aprimoramentos no editor e suporte a virtualização.

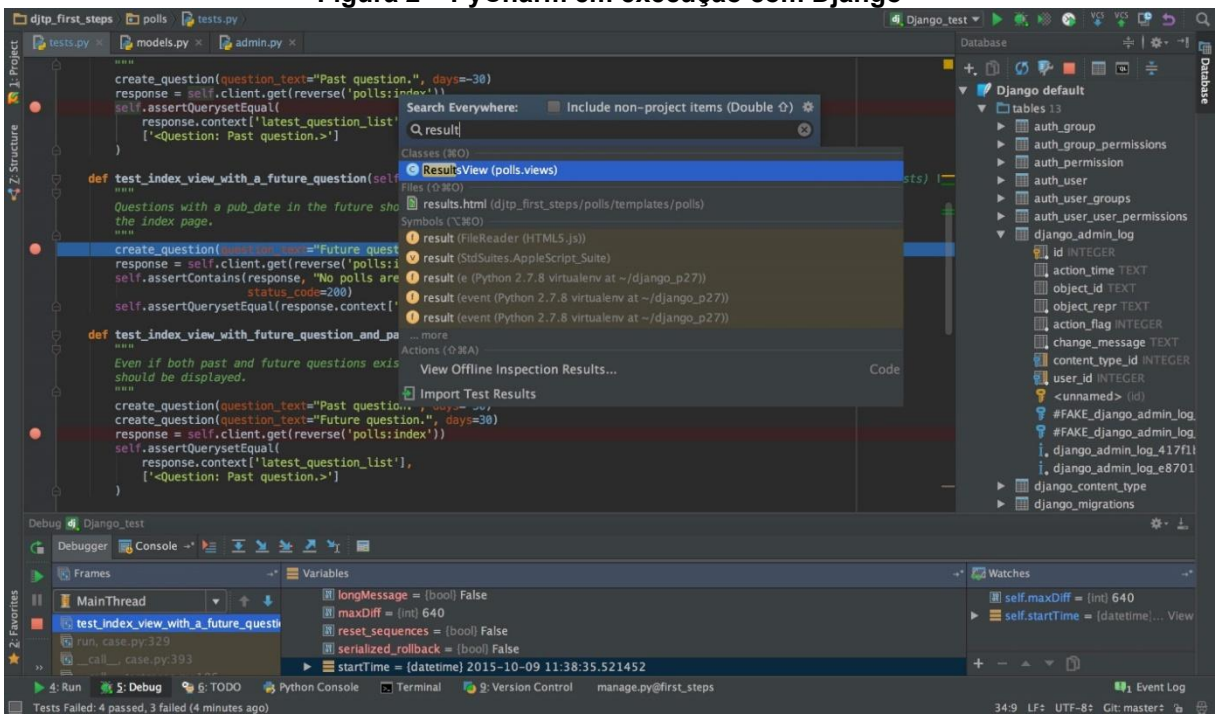
3.3.1 Desenvolvimento web

Para desenvolvimento WEB, o editor de código inteligente do PyCharm fornece suporte de primeira classe para Python, JavaScript, CoffeeScript, TypeScript e CSS (*Cascading Style Sheets*).

Para desenvolvimento WEB utilizando Python, o PyCharm oferece suporte aos mais modernos frameworks como Django, Flask, Google App Engine, Pyramid e Web2py. Ainda permite a visualização e edição em tempo real que permite que uma página seja aberta no editor e no navegador sendo que as mudanças são aplicadas instantaneamente (JETBRAINS, 2000).

Na Figura 2, tem-se a captura de tela da IDE Pycharm em execução com Django.

Figura 2 – PyCharm em execução com Django



Fonte: JetBrains (2020).

3.4 Visual studio code (vs code)

O *Visual Studio Code* (VS Code) é um editor de código-fonte desenvolvido e distribuído pela Microsoft. O código fonte é aberto a todos sob a MIT *license* e disponível no GitHub, porém a aplicação em si é distribuída sob a Microsoft *product license* tradicional de maneira gratuita. O editor é extremamente popular sendo que em uma pesquisa do site *overflow* mais de 71% dos respondentes usam o VS Code e atualmente existem mais de 30 mil plugins disponíveis (BRUDO, 2021).

3.5 Django

Django é uma aplicação gratuita e de código aberto para desenvolvimento Web e faz com que as tarefas sejam executadas de forma mais rápida e segura.

A ferramenta facilita o desenvolvimento, cuidando das tarefas de autenticação dos usuários, administração de conteúdo, mapas de site, feed RSS, entre outras.

A segurança implementada na ferramenta faz com que os erros mais comuns sejam evitados, tais como: Injeção de SQL, scripts entre sites, falsificações de solicitações e *clickjacking*.

A escalabilidade é outro ponto a ser salientado, pois o Django se adapta rapidamente às demandas de tráfego mais pesado.

Sua versatilidade faz com que várias empresas o utilizem para construção de vários tipos de aplicações como sistemas de gerenciamento de conteúdo para redes sociais ou plataformas de comunicação científica.

Algumas das características fundamentais da ferramenta são (DJANGO, 2014):

- Mapeador relacional de objetos: Definição de modelos de dados inteiramente em Python com a obtenção de uma API (*Application Programming Interface*, ou Interface de Programação de Aplicação) rica e dinâmica e acesso ao banco de dados de forma gratuita;
- URLs e visualizações: Design mais limpo para as URLs das aplicações, sem a necessidade de inclusão de termos como ".php" ou ".asp";
- Modelos: Equilíbrio entre personalização e facilidade, foi projetada para que se tenha uma facilidade de aprendizado para usuários acostumados com HTML, mas com nível de personalização para que programadores alterem o código para que se adeque às suas necessidades;
- Formulários: Biblioteca poderosa que renderiza formulários em HTML e os converte em dados nativos do Python. Também há a possibilidade de geração de formulários a partir dos modelos existentes na ferramenta;
- Autenticação: Sistema de autenticação capaz de lidar com usuários, grupos, permissões e sessões de usuários baseadas em *cookies*. Traz facilidade e segurança para a criação de aplicações que necessitem de criação de contas, *login* e *logout*;
- Administração: A *interface* de administração é criada automaticamente a partir da leitura de metadados dos modelos e, a partir daí, pode ser usada para o gerenciamento do conteúdo do *site* ou aplicação;
- Internacionalização: Suporte para a tradução de texto para diferentes idiomas, formatação de data e hora, números e fusos horários de cada região. Permite que o desenvolvedor especifique qual a parte do site poderá ser traduzida e/ou formatada pelo usuário;

- Segurança no Django contra: *clickjacking*, *cross-site scripting*, *Cross Site Request Forgery* (CSRF), Injeção SQL e Execução remota de código.

3.6 Angular

Angular é uma *web application framework*[1] desenvolvido pelo time Angular dentro da Google que utiliza a linguagem TypeScript como base. O Angular foi projetado para ser dimensionado de um projeto pequeno desenvolvido por uma pessoa até grandes projetos em escala corporativa.

O framework Angular inclui (ANGULAR, 2021):

- Uma estrutura baseada em componentes para a construção de aplicativos da web escalonáveis;
- Uma coleção de bibliotecas bem integradas que cobrem uma ampla variedade de recursos, incluindo roteamento, gerenciamento de formulários, comunicação cliente-servidor e muito mais;
- Um pacote de ferramentas de desenvolvedor para ajudá-lo a desenvolver, construir, testar e atualizar seu código.

Para inicializar, desenvolver, estruturar e manter aplicativos em Angular existe a Angular CLI (*Command-Line Interface*, ou Interface de Linha de Comando), ou seja, você consegue fazer a funções acima direto da *shell* de comando (ANGULAR, 2021).

3.7 Celery task e celery scheduler

Celery é um sistema distribuído, simples flexível e confiável para gerenciamento e processamento de grandes quantidades de mensagens. Além disso, é uma ferramenta de código aberto.

---

[1] *Framework* é conjunto de bibliotecas ou componentes utilizados para auxiliar o desenvolvimento de uma aplicação.

3.7.1 Tarefas

Tarefas são os blocos de construção dos aplicativos Celery. Ela tem a função dupla, sendo que define o que acontece quando uma tarefa é chamada e o que acontece quando um processo recebe essa mensagem.

Uma mensagem de tarefa desaparece somente quando é confirmada por um processo, sendo que esses podem reservar mensagens com antecedência e, mesmo que o processo for finalizado por algum erro, a mensagem será reenviada para outro processo.

Idealmente, as mensagens de tarefas não devem apresentar erros mesmo que sejam chamadas várias vezes com os mesmos argumentos. Cada processo deve reconhecer a mensagem com antecedência, antes de ser executada, para que uma tarefa já iniciada não seja executada novamente.

3.7.2 Nomes

Cada tarefa deve ter um nome único que deve ser gerado a partir do nome da função caso o nome personalizado não seja fornecido. A prática ideal é usar o nome módulo.

3.7.3 Logging

Logging é o processo que irá configurar o registro automaticamente ou, caso necessários, pode ser configurado manualmente. Um *logger* especial é disponibilizado e pode ser herdado automaticamente o nome da tarefa e a ID exclusiva.

3.7.4 Resultados

Os resultados podem ser acompanhas no Aipo por meio do estado atual das tarefas que contém o resultado de uma tarefa bem-sucedida ou os dados necessários para o rastreamento e correção de uma falha.

No período de sua duração a tarefa passará por vários estados possíveis, sendo que cada estado pode ter metadados associados a ele. No momento que a tarefa passa por um novo estado, o anterior é esquecido.

3.7.5 Back-ends de resultados

O Celery deve enviar os *back-ends* de resultados para algum lugar para que as tarefas possam ser acompanhadas e recuperadas posteriormente. Existem várias opções para *back-end* de resultados, como: SQLAlchemy, Django ORM, Memcached, RabbitMQ, QPid (RPC), MongoDB e Redis.

3.7.6 Back-end do resultado do banco de dados

A manutenção dos estados no banco de dados pode ser conveniente, principalmente para aplicações WEB com banco de dados já implementados, mas apresenta algumas limitações como: custo e isolamento de transações (que não são adequados para dados que sofrem mudanças constantes).

3.8 Rabbit mq

Rabbit MQ é um corretor de mensagens de código aberto que pode ser implementado localmente ou em nuvem. Dentre seus recursos estão (RABBITMQ, 2011):

- Mensagens assíncronas: Suporte a vários protocolos de mensagens, enfileiramento, confirmação de entrega ao destinatário, roteamento flexível de filas, troca múltipla;
- Compatibilidade: Suporte a várias linguagens de programação como: Java, .NET, PHP, Python, Javascript, Ruby, entre outras;
- Segurança: Autenticação, autorização, suporte a *Transport Layer Security* (TLS) e *Lightweight Directory Access Protocol* (LDAP). Pode ser implementado em nuvens públicas ou privadas;
- Monitoramento: Possui ferramenta em linha de comando e com interface de usuário para gerenciamento e monitoramento de mensagens.

A ferramenta utiliza o *Advanced Message Queuing Protocol* (AMQP) que é um protocolo de comunicação em rede responsável pelo roteamento de distribuição de mensagens de aplicações.

3.8.1 Filas

Existem diversos padrões de filas em programação, mas em todos os casos tem-se ao menos os três elementos (SILVA, 2019):

- *producing*: Aplicação responsável pela produção da mensagem;
- *queue*: A fila que fica dentro do Rabbit MQ e aloca as mensagens enviadas pela aplicação. Geralmente o limite dessa fila é o espaço em disco no servidor ou sua capacidade de memória;
- *consuming*: Geralmente é uma aplicação que recebe e processa a mensagem alocada na fila.

No Rabbit MQ tem-se cinco padrões de implementação de filas (SILVA, 2019):

- *work queues*: Utilizado em processos mais demorados nos quais as tarefes são colocadas em fila para serem executadas posteriormente, liberando o usuário do sistema para outras tarefas;
- *publish/subscribe*: Basicamente igual padrão anterior com a diferença que neste são utilizadas diversas filas ao mesmo. O usuário não pode controlar a qual fila a mensagem será alocada;
- *routing*: Igualmente ao anterior, esse padrão trabalha com diversas filas ao mesmo tempo, mas com a função de o usuário poder alocar a mensagem na fila de sua preferência;
- *topics*: Nesse padrão a mensagem pode ser alocada em diversas filas simultaneamente e processada de diferentes formas. Isso só é possível com uma lista de palavras-chave que classificam a mensagem em filas diferentes;
- *Remote Procedure Call* (RPC): É utilizado para comunicação entre aplicações diferentes. As mensagens são alocadas nas filas e somente são lidas pelo servidor quando ele estiver disponível. Caso ocorra algum problema de comunicação, o cliente recebe uma mensagem de erro.

3.9 Hiper text markup language (html)

A *Hiper Text Markup Language* (HTML, ou Linguagem de Marcação de Hipertexto) é a linguagem mais básica de programação web. Com ela podemos definir as estruturas de todo o conteúdo web.

Além do HTML, outras ferramentas são utilizadas para uma melhor apresentação do conteúdo web, a exemplo do CSS (*Cascading Style Sheets*) ou linguagens que implementam funcionalidades, a exemplo do JavaScript.

O termo “hipertexto” faz referência aos links que são a base de toda comunicação web, pois conectam uma página a outra ou a alguma outra funcionalidade dentro da mesma página.

Já o termo “marcação” é relacionado a funcionalidade de inclusão de conteúdo dentro de uma página como cabeçalho, título, corpo, imagens etc.

Os elementos HTML são separados de outros textos em documentos por *tags* (nome do elemento entre os caracteres “<” e “>”). Tais elementos não possuem diferenciação entre letras maiúsculas e minúsculas, sendo aceito qualquer forma de escrita.

A linguagem HTML é muito versátil e há maneiras de incorporar diversos recursos com formulários, imagens, áudios, vídeos, tabelas, referências etc.

XHTML: Trata-se de uma variação do HTML que utiliza a sintaxe do XML (*eXtensible Markup Language*) com a qual pode-se utilizar os elementos específicos de XML e outras ferramentas específicas para XML pode ser utilizada na elaboração de páginas (MOZILLA, 2012).

3.10 Cascading style sheets (css)

O *Cascading Style Sheets* (CSS) é uma linguagem da folha de estilo utilizada para descrever a apresentação de um documento escrito em HTML ou em XML (incluindo várias linguagens em XML como SVG, MathML ou XHTML). O CSS descreve como elementos são mostrados na tela, no papel, na fala ou em outras mídias. CSS é mantido pela *World Wide Web Consortium* (W3C) desde a sua primeira versão (*W3C CSS Recommendation* (CSS1)) lançada em 1996 (MOZILLA, 2016).

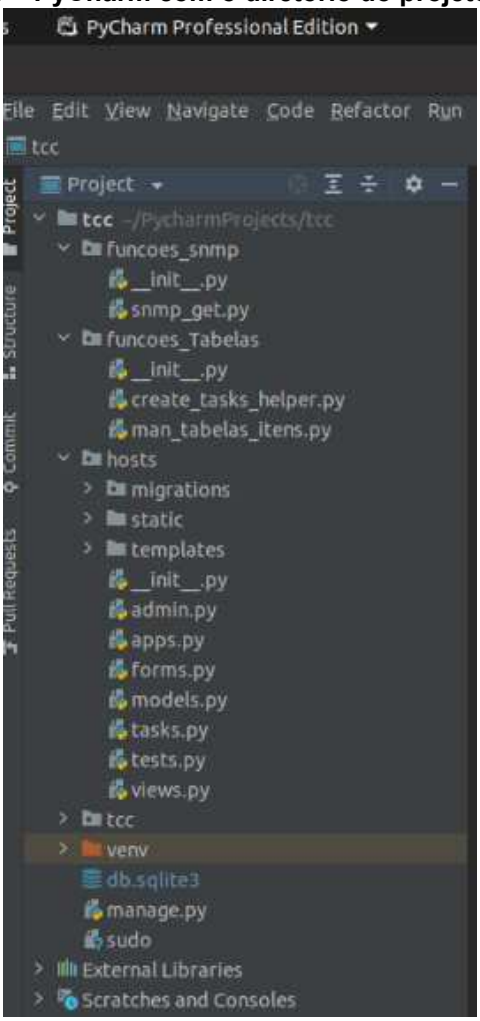
4 APLICAÇÃO DESENVOLVIDA

Esta seção e suas subseções trazem detalhes do desenvolvimento do programa de monitoramento de rede. Descrevendo detalhadamente as bibliotecas da linguagem de programação Python que que foram utilizadas e a organização dos elementos componentes dessa aplicação.

4.1 Back-end

O *back-end* do projeto realizado tem como objetivo o monitoramento de dispositivos de rede via protocolo SNMP. Para isso foi utilizado o *framework* Django e na Figura 3, pode-se observar na IDE PyCharm as principais pastas e arquivos do projeto.

Figura 3 – PyCharm com o diretório do projeto aberto



Fonte: Autoria própria (2022).

4.1.1 Diretório “funções_snmp”

O diretório “funções_snmp” é uma pasta criada manualmente no projeto que possui dois arquivos com extensão .py, são eles:

- **init__.py**: Conforme documentação da seção *Modules* disponível no site “docs.python.org”, esse arquivo é utilizado para o interpretador Python identificar o diretório “funções_snmp” como um módulo Python e com isso todas as funções, variáveis, objetos, entre outros; contidos nos arquivos desse módulo podem ser importados em outros arquivos e diretórios do projeto utilizando o comando *from* e *import*. Esse arquivo é criado automaticamente pelo ambiente de desenvolvimento PyCharm e estará presente em vários outros diretórios do projeto.
- **snmp_get.py**: Arquivo criado manualmente que contém umas das principais funções do projeto, a snmpGet. Essa função recebe dois argumentos principais sendo eles o *Internet Protocol* (IP) e *Object IDentifier* (OID). O objetivo dela é realizar um snmp-request no IP de destino informado buscando o valor da variável na *Management Information Bases* (MIB) do dispositivo.

A seguir tem-se as variáveis importantes do arquivo “snmp_get.py”, apresentado no Quadro 2, e suas respectivas finalidades:

- **ip:** endereço IP do dispositivo na rede ao qual será feito o snmpGet, deve ser passado o parâmetro no formato texto.
- **oid:** Endereço do objeto na tabela MIB ao qual se quer capturar o valor e deve ser passada no formato texto para função.
- **community:** se não informada na chamada da função é assumido que o valor da variável é a ‘*public*’ no formato texto que é a *community* padrão do protocolo SNMP.
- **versão_snmp:** se não informada a variável, a função assume que o valor é 0 (zero). Conforme documentação da biblioteca “pysnmp” disponível no site “pysnmp.readthedocs.io” na página *SNMP Versions* (ETINGOF, 2020). Quando passado o valor 0 (zero) ao segundo parâmetro da função *CommunityData* contido no arquivo *auth.py* da biblioteca “pynsmp”, o “snmpGet” utiliza a versão 1 (um) do protocolo SNMP. O projeto foi

desenvolvido para funcionar utilizando somente a versão 1 (um) do protocolo SNMP, ou seja, o valor da variável “versao_snmp” nunca é alterado no código.

- **porta:** se não informada a variável a função assumi que o valor é 161 e é utilizada na classe *UdpTransportTarget* para montar o *snmp-request* que será realizado no dispositivo da rede que se quer monitorar. Soares *et al.* (2010), afirmam que normalmente é utilizada a porta UDP 161 no agente para o recebimento do *snmp-request* e 162 para o gerente receber a resposta.
- **errorIndication:** é retornada quando ocorre erros localmente no Gerente da arquitetura SNMP quando realizado o *snmp-request*, como por exemplo *time-outs* ou configurações locais erradas.
- **errorStatus:** é retornado quando ocorre um erro no agente da arquitetura SNMP quando realizado o *snmp-request*, como por exemplo um endereço OID que não existe ou não está acessível;
- **varBinds:** Essa variável é uma lista que retorna o resultado caso o *snmp-request* tenha sido realizado com sucesso. Ele traz o valor da variável capturada no agente.

Quadro 2 – Código-fonte Python: snmp_get.py

```
from pysnmp.hlapi import getCmd, SnmpEngine, CommunityData,
UdpTransportTarget, ContextData, ObjectType, ObjectIdentity

# SnmpEngiene() e o objeto central da operaçao e serve como identificador
# que e atribuido automaticamente nesse exemplo
# comunnity e versao do SNMP para enviar no SNMP Request
# ip e porta de destino para o SNMP Request. Ele é um parâmetro do
# utilizado somente no SNMPv3 e deve ser testado sua remoção desse projeto
# OID que é passada a classe ObjectIdentity para identificar a OID (se
# necessário) e ao objeto Object Type para criar uma tupla de valores.

def snmpGet(ip, oid, community='public', versao_snmp=0, porta=161):
    errorIndication, errorStatus, errorIndex, varBinds = next(
getCmd(SnmpEngine(), CommunityData(community, mpModel=versao_snmp),
UdpTransportTarget((ip, porta)), ContextData(),
               ObjectType(ObjectIdentity(oid))))
    if errorIndication:  # Retornado quando ocorre erro no servidor local
        return (str(errorIndication))
    elif errorStatus:  # Retornado quando ocorre erro no agente
        return ('%s at %s' % (errorStatus.prettyPrint(),
          errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
    else:
        for varBind in varBinds:
            return str(varBind).split(' = ')[1] # não ocorre erro
```

Fonte: Autoria própria (2022).

4.1.2 Diretório “funções_tabelas”

O diretório “funções_Tabelas”, assim como no diretório “funções_snmp”, possui um arquivo chamado __init__.py para identificar o diretório como um módulo Python e poder ser utilizado nos demais diretórios do projeto. Nesse mesmo diretório existem mais dois arquivos chamados *create_tasks_helper.py* e *man_tabelas_itens.py* que foram criados manualmente para o projeto.

4.1.3 Arquivo de Código-fonte: create_tasks_helper.py

O arquivo “create_tasks_helper.py”, como o nome mesmo já indica, possui funções de auxílio a criação de tarefas no projeto. Essas tarefas nada mais são do que funções que são executadas assincronamente a execução do código do projeto, ou seja, quando chamadas ocorrem em paralelo a execução do projeto e não precisam ser esperados o retorno delas para a execução do código principal.

4.1.4 Arquivo de código-fonte: man_tabelas_itens.py

O arquivo “man_tabelas_itens.py”, possui funções que lidam diretamente com o banco de dados da aplicação como consultas de *itens*, *templates*, *hosts* e criação de novas tabelas. Nesse arquivo possui também linguagem de programação *Structured Query Language* (SQL) que pode ser utilizada em um arquivo Python com a utilização de três aspas duplas.

4.1.5 Diretório “hosts”

O Diretório “hosts” é o diretório da aplicação chamada *Hosts* dentro do projeto. Dentro do diretório de uma aplicação de um projeto Django constam arquivos referente aquela aplicação como telas, configurações e toda a codificação referente aquela aplicação do projeto. Os principais arquivos desse diretório para o projeto são o *models.py* e *tasks.py*.

O termo aplicação descreve um pacote Python que disponibiliza várias funcionalidades a um projeto. Aplicações podem ser reutilizadas em vários projetos diferentes. Aplicações contém várias combinações de *models*, *views*, *templates tags*, *static files*, URLSs, *middlewares*, entre outros. Normalmente eles são atrelados a um projeto por meio do parâmetro INSTALLED_APPS e opcionalmente com outros

mecanismos como URLconfs, configurações de MIDDLEWARE ou herança de templates. É importante entender que um aplicativo Django é um conjunto de código que interage com várias partes da estrutura e está contido dentro do projeto Django (DJANGO, 2008).

4.1.6 Arquivo de código-fonte: models.py

O código-fonte do arquivo “models.py”, apresentado no Quadro 3, é uma classe do projeto Django que auxilia o desenvolvedor a criar e gerenciar as tabelas e comunicação com o banco de dados. Nesse arquivo podem ser criados objetos que herdam de *models.py* e gerenciam operações de inserção, leitura, atualização e exclusão de registros no banco de dados.

Quadro 3 – Código-fonte Python: models.py

(continua)

```
from django.db import models

# Lista de formatos de informação retornada pelo snmp_get.py
TIPOS_DE_INFORMACAO_RETORNADA_SNMP_GET = [
    ('NI', 'Numérico(Inteiro)'),
    ('ND', 'Numérico(Decimal)'),
    ('CH', 'Caracter'),
    ('LG', 'Log'),
    #('TX', 'Texto')
]


# Lista de unidades do intervado de atualização
UNIDADES_INTERVALO_ATUALIZACAO = [
    ('seconds', 'Segundos'),
    ('minutes', 'Minutos'),
    ('hours', 'Horas'),
    ('days', 'Dias'),
    ('months', 'Meses'),
    ('years', 'Anos')
]


# Lista de unidades do intervado de atualização
UNIDADES_INTERVALO_ARMAZENAMENTO = [
    ('minutes', 'Minutos'),
    ('hours', 'Horas'),
    ('days', 'Dias'),
    ('months', 'Meses'),
    ('years', 'Anos')
]
```

Quadro 3 – Código-fonte Python: models.py

(conclusão)

```
# Classe Model que gerencia a tabela Item do banco de dados
class Item(models.Model):
    item_nome = models.CharField(max_length=100)
    item_oid = models.CharField(max_length=100)
    item_tipoInformacao =
models.CharField(choices=TIPOS_DE_INFORMACAO_RETORNADA_SNMP_GET,
max_length=100)
    item_intervaloAtualizacao =
models.PositiveSmallIntegerField(default=1)
    item_intervaloAtualizacaoUn =
models.CharField(choices=UNIDADES_INTERVALO_ATUALIZACAO, max_length=100,

default=UNIDADES_INTERVALO_ATUALIZACAO.__getitem__(1))
    item_tempoArmazenamentoDados =
models.PositiveSmallIntegerField(default=90)
    item_tempoArmazenamentoDadosUn =
models.CharField(choices=UNIDADES_INTERVALO_ARMAZENAMENTO, max_length=100,

default=UNIDADES_INTERVALO_ARMAZENAMENTO.__getitem__(2))
    item_expressaoConversao = models.CharField(max_length=100, null=True,
blank=True)

    # Função que retorna na view o nome do objeto Item
    def __str__(self):
        return self.item_nome

    # Classe que retorna o nome correto da classe Item no plural
    class Meta:
        verbose_name_plural = 'Itens'



# Classe Model que gerencia a tabela Template do banco de dados
class Template(models.Model):
    template_nome = models.CharField(max_length=100)
    template_item = models.ManyToManyField(Item)
    template_observacoes = models.TextField(null=True, blank=True)

    # Função que retorna na view o nome do objeto Template
    def __str__(self):
        return self.template_nome

# Classe Model que gerencia a tabela Host do banco de dados
class Host(models.Model):
    host_nome = models.CharField(max_length=100)
    host_nomeTabela_snmpGet = models.CharField(max_length=100)
    host_ip = models.GenericIPAddressField(protocol='IPv4',
unpack_ipv4=False)
    host_porta = models.PositiveIntegerField(default=161)
    host_template = models.ManyToManyField(Template)
    host_observacoes = models.TextField(null=True, blank=True)
    host_status = models.BooleanField(default=True)
    host_community = models.CharField(max_length=32, default='public')

    # Função que retorna na view o nome do objeto Host
    def __str__(self):
        return self.host_nome
```

Fonte: Autoria própria (2022).

4.1.7 Arquivo de código-fonte: tasks.py

O arquivo de código-fonte “tasks.py”, apresentado no Quadro 4, é onde fica armazenado as funções assíncronas do projeto. Cada função é criada com a chamada do *decorator* @shared_task. O *decorator* @shared_task torna utilizável a função em todos as aplicações de um projeto sem a necessidade de criar uma instância concreta de uma aplicação (DJANGO, 2021).

Um exemplo de chamada de funções desse arquivo é a função *task_snmp_get* que é chamada assincronamente através da biblioteca *celery beat* que realiza a varredura dos registros cadastrados na tabela *django_celery_beat_periodictask* que por sua vez armazena as chamadas da função *task_snmp_get* com seus respectivos parâmetros.

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
# Create your tasks here
from __future__ import absolute_import, unicode_literals

from celery import shared_task

from funcoes_snmp.snmp_get import snmpGet

from       funcoes_Tabelas.create_tasks_helper       import       getItens,
createTaskSnmpGet, createTaskCleanData
from    funcoes_Tabelas.man_tabelas_itens    import    insertSnmpGetResult,
clean_data

from django_celery_beat.models import PeriodicTask, IntervalSchedule

from .models import Host, Item, Template

import time

@shared_task
def    task_snmp_get(host_nomeTabela_snmpGet,    host_community,    id_item,
nome_item, ip, oid, porta, template_ids):
    info = snmpGet(ip=ip, oid=oid, porta=porta, community=host_community)
    insertSnmpGetResult(host_nomeTabela_snmpGet, id_item, nome_item, info)
    return info


@shared_task
def create_task_snmpGet_host_created(host_nomeTabela_snmpGet,
                                     host_ip,
                                     host_porta,
                                     host_status,
                                     host_community):
    itens = getItens(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                     host_ip=host_ip,
                     host_porta=host_porta,
                     host_status=host_status)
```

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
for item in itens:
        templates = Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                                            host__host_ip=host_ip,
                                            host__host_porta=host_porta,
                                            host__host_status=host_status)

        createTaskSnmpGet(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                          host_ip=host_ip,
                          host_porta=host_porta,
                          host_community=host_community,
                          templates=templates,
                          item_id=item.id,
                          item_nome=item.item_nome,
                          item_oid=item.item_oid,

item_intervaloAtualizacao=item.item_intervaloAtualizacao,

item_intervaloAtualizacaoUn=item.item_intervaloAtualizacaoUn,)



@shared_task
def create_task_snmpGet_host_updated(host_nomeTabela_snmpGet,
                                     host_ip,
                                     host_porta,
                                     host_status,
                                     host_community):
    lista_PeriodicTasks_cadastradas                                          =
PeriodicTask.objects.filter(name__contains=('SNMPGETTASK='                   +
host_nomeTabela_snmpGet))

    for periodicTask in lista_PeriodicTasks_cadastradas:
        try:
            periodicTask.delete()
        except Exception as e:
            print(str(e.args[0]))
            time.sleep(4)
            periodicTask.delete()

    if host_status:
        itens = getItens(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                         host_ip=host_ip,
                         host_porta=host_porta,
                         host_status=host_status)

        for item in itens:
            templates = Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                                                host__host_ip=host_ip,

host__host_porta=host_porta,

host__host_status=host_status)
```

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
createTaskSnmpGet(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                              host_ip=host_ip,
                              host_porta=host_porta,
                              host_community=host_community,
                              templates=templates,
                              item_id=item.id,
                              item_nome=item.item_nome,
                              item_oid=item.item_oid,

item_intervaloAtualizacao=item.item_intervaloAtualizacao,

item_intervaloAtualizacaoUn=item.item_intervaloAtualizacaoUn)


@shared_task
def create_task_snmpGet_template_updated(template_id, template_nome):
    hosts                                                                    =
Host.objects.filter(host_template__template_nome=template_nome,
                                host_template__id=template_id)

    for host in hosts:
        lista_PeriodicTasks_cadastradas                                      =
PeriodicTask.objects.filter(name__contains=('SNMPGETTASK='                   +
host.host_nomeTabela_snmpGet))

        for periodicTask in lista_PeriodicTasks_cadastradas:
            periodicTask.delete()

        if host.host_status:
            itens                                                            =
getItens(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                              host_ip=host.host_ip,
                              host_porta=host.host_porta,
                              host_status=host.host_status)

            for item in itens:
                templates                                                    =
Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,

host__host_ip=host.host_ip,

host__host_porta=host.host_porta,

host__host_status=host.host_status)

createTaskSnmpGet(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                                    host_ip=host.host_ip,
                                    host_porta=host.host_porta,
                                    host_community=host.host_community,
                                    templates=templates,
                                    item_id=item.id,
                                    item_nome=item.item_nome,
                                    item_oid=item.item_oid,
item_intervaloAtualizacao=item.item_intervaloAtualizacao,
item_intervaloAtualizacaoUn=item.item_intervaloAtualizacaoUn)
```

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
@shared_task
def create_task_snmpGet_template_deleted(hosts_ids):
    for host_id in hosts_ids:
        hosts = Host.objects.filter(id=host_id)

    for host in hosts:
        lista_PeriodicTasks_cadastradas                                    =
PeriodicTask.objects.filter(name__contains=('SNMPGETTASK='                 +
host.host_nomeTabela_snmpGet))
        print(lista_PeriodicTasks_cadastradas)

        for periodicTask in lista_PeriodicTasks_cadastradas:
            periodicTask.delete()

        if host.host_status:
            itens                                                          =
getItens(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                             host_ip=host.host_ip,
                             host_porta=host.host_porta,
                             host_status=host.host_status)

            for item in itens:
                templates                                                  =
Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,

host__host_ip=host.host_ip,

host__host_porta=host.host_porta,

host__host_status=host.host_status)


createTaskSnmpGet(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                                  host_ip=host.host_ip,
                                  host_porta=host.host_porta,
                                  host_community=host.host_community,
                                  templates=templates,
                                  item_id=item.id,
                                  item_nome=item.item_nome,
                                  item_oid=item.item_oid,

item_intervaloAtualizacao=item.item_intervaloAtualizacao,

item_intervaloAtualizacaoUn=item.item_intervaloAtualizacaoUn)


@shared_task
def create_task_snmpGet_item_updated(item_id,
                                     item_nome_old,
                                     item_oid_old,
                                     item_intervaloAtualizacao_old,
                                     item_intervaloAtualizacaoUn_old):
    lista_periodicTasks =
PeriodicTask.objects.filter(name__contains=('_item_id:' +
str(item_id))).filter(name__contains=('SNMPGETTASK'))
```

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
    item = Item.objects.get(id=item_id)

    for periodicTask in lista_periodicTasks:
        args = periodicTask.args
        print(item_nome_old, item_oid_old)
        args   =   args.replace('"'   +   item_nome_old   +   '"',   '"'   +
item.item_nome + '"')
        args = args.replace('"' + item_oid_old + '"', '"' + item.item_oid
+ '"')
        periodicTask.args = args

        intervalo_id = None

        novoIntervalo                                                    =
IntervalSchedule(every=item_intervaloAtualizacao_old,

period=item_intervaloAtualizacaoUn_old)

        intervalosCadastrados = IntervalSchedule.objects.all()

        for intervalo in intervalosCadastrados:
            if     (intervalo.every     ==     novoIntervalo.every)     and
(intervalo.period == novoIntervalo.period):
                intervalo_id = intervalo.id
                print('ACHEI UM INTERVALO NO BD')

        if intervalo_id is None:
            novoIntervalo.save()
            intervalo_id = novoIntervalo.id
            print('NÃO ACHEI UM INTERVALO E SALVEI NO BD')

        periodicTask.interval_id = intervalo_id

        periodicTask.save()


# =============================================================================
@shared_task
def task_clean_data(item_id,
                    item_tempoArmazenamentoDados,
                    item_tempoArmazenamentoDadosUn,
                    host_nomeTabela_snmpGet):
    clean_data(item_id=item_id,
               item_tempoArmazenamentoDados=item_tempoArmazenamentoDados,

item_tempoArmazenamentoDadosUn=item_tempoArmazenamentoDadosUn,
               host_nomeTabela_snmpGet=host_nomeTabela_snmpGet)

@shared_task
def create_task_CleanData_host_created(host_nomeTabela_snmpGet,
                                       host_ip,
                                       host_porta,
                                       host_status):
    itens = getItens(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                     host_ip=host_ip,
                     host_porta=host_porta,
                     host_status=host_status)
```

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
    for item in itens:
        templates = Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                                            host__host_ip=host_ip,
                                            host__host_porta=host_porta,
                                            host__host_status=host_status)


createTaskCleanData(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                            templates=templates,
                            item_id=item.id,

item_tempoArmazenamentoDados=item.item_tempoArmazenamentoDados,

item_tempoArmazenamentoDadosUn=item.item_tempoArmazenamentoDadosUn)

@shared_task
def create_task_CleanData_host_updated(host_nomeTabela_snmpGet,
                                       host_ip,
                                       host_porta,
                                       host_status):
    lista_PeriodicTasks_cadastradas                                    =
PeriodicTask.objects.filter(name__contains=('CLEANDATATASK='           +
host_nomeTabela_snmpGet))

    for periodicTask in lista_PeriodicTasks_cadastradas:
        try:
            periodicTask.delete()
        except Exception as e:
            print(str(e.args[0]))
            time.sleep(4)
            periodicTask.delete()

    if host_status:
        itens = getItens(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                         host_ip=host_ip,
                         host_porta=host_porta,
                         host_status=host_status)

        for item in itens:
            templates = Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                                                host__host_ip=host_ip,

host__host_porta=host_porta,

host__host_status=host_status)


createTaskCleanData(host_nomeTabela_snmpGet=host_nomeTabela_snmpGet,
                                templates=templates,
                                item_id=item.id,

item_tempoArmazenamentoDados=item.item_tempoArmazenamentoDados,

item_tempoArmazenamentoDadosUn=item.item_tempoArmazenamentoDadosUn)
```

Quadro 4 – Código-fonte Python: tasks.py

(continua)

```
@shared_task
def create_task_CleanData_template_updated(template_id, template_nome):
    hosts                                                                =
Host.objects.filter(host_template__template_nome=template_nome,
                                host_template__id=template_id)
    for host in hosts:
        lista_PeriodicTasks_cadastradas                                  =
PeriodicTask.objects.filter(name__contains=('CLEANDATATASK='             +
host.host_nomeTabela_snmpGet))

        for periodicTask in lista_PeriodicTasks_cadastradas:
            periodicTask.delete()
        if host.host_status:
            itens                                                        =
getItens(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                             host_ip=host.host_ip,
                             host_porta=host.host_porta,
                             host_status=host.host_status)
            for item in itens:
                templates                                                =
Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
  host__host_ip=host.host_ip,
  host__host_porta=host.host_porta,
  host__host_status=host.host_status)


createTaskCleanData(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                                    templates=templates,
                                    item_id=item.id,

item_tempoArmazenamentoDados=item.item_tempoArmazenamentoDados,

item_tempoArmazenamentoDadosUn=item.item_tempoArmazenamentoDadosUn)


@shared_task
def create_task_CleanData_template_deleted(hosts_ids):
    for host_id in hosts_ids:
        hosts = Host.objects.filter(id=host_id)

    for host in hosts:
        lista_PeriodicTasks_cadastradas                                  =
PeriodicTask.objects.filter(name__contains=('CLEANDATATASK='             +
host.host_nomeTabela_snmpGet))
        print(lista_PeriodicTasks_cadastradas)

        for periodicTask in lista_PeriodicTasks_cadastradas:
            periodicTask.delete()

        if host.host_status:
            itens                                                        =
getItens(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                             host_ip=host.host_ip,
                             host_porta=host.host_porta,
                             host_status=host.host_status)
```

Quadro 4 – Código-fonte Python: tasks.py

(conclusão)

```
            for item in itens:
                templates                                                    =
Template.objects.filter(template_item__id=item.id,

host__host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,

host__host_ip=host.host_ip,

host__host_porta=host.host_porta,

host__host_status=host.host_status)

createTaskCleanData(host_nomeTabela_snmpGet=host.host_nomeTabela_snmpGet,
                                    templates=templates,
                                    item_id=item.id,

item_tempoArmazenamentoDados=item.item_tempoArmazenamentoDados,

item_tempoArmazenamentoDadosUn=item.item_tempoArmazenamentoDadosUn)

@shared_task
def create_task_CleanData_item_updated(item_id,
                                     item_tempoArmazenamentoDados_old,
                                     item_tempoArmazenamentoDadosUn_old):
    lista_periodicTasks                                                      =
PeriodicTask.objects.filter(name__contains=('_item_id:'                      +
str(item_id))).filter(name__contains=('CLEANDATATASK'))
    item = Item.objects.get(id=item_id)

    for periodicTask in lista_periodicTasks:
        args = periodicTask.args
        args = args.replace('"' + str(item_tempoArmazenamentoDados_old) +
'"', ('"' + str(item.item_tempoArmazenamentoDados) + '"'))
        args  =  args.replace('"'  +  item_tempoArmazenamentoDadosUn_old  +
'"', '"' + item.item_tempoArmazenamentoDadosUn + '"')
        periodicTask.args = args

        intervalo_id = None

        novoIntervalo = IntervalSchedule(every=5,
                                         period='minutes')

        intervalosCadastrados = IntervalSchedule.objects.all()

        for intervalo in intervalosCadastrados:
            if     (intervalo.every     ==     novoIntervalo.every)     and
(intervalo.period == novoIntervalo.period):
                intervalo_id = intervalo.id
                print('ACHEI UM INTERVALO NO BD')

        if intervalo_id is None:
            novoIntervalo.save()
            intervalo_id = novoIntervalo.id
            print('NÃO ACHEI UM INTERVALO E SALVEI NO BD')

        periodicTask.interval_id = intervalo_id

        periodicTask.save(
```

Fonte: Autoria própria (2022).

4.1.8 Diretório “tcc”

O “tcc” é o diretório do projeto geral. Nele constam arquivos de configuração geral do projeto como *Uniform Resource Locators* (URL), banco de dados, servidor e configuração do Celery que é a biblioteca principal que gerencia as tarefas assíncronas e periódicas como o snmp-requests por exemplo.

O termo projeto descreve aplicação web Django como um todo. O projeto em si contém arquivos de configuração geral. Por exemplo, quando é executado os comandos django-admin, startproject mysite está sendo executado arquivos do projeto principal Django (DJANGO, 2021).

4.1.9 Diretório “venv”

Conforme seção “venv – Criação de Ambientes Virtuais” disponível no site “docs.python.org” (PYTHON, 2020), o diretório “venv” é onde fica armazenado os arquivos e código referentes ao ambiente virtual responsável por executar o projeto.

Um ambiente virtual é onde fica armazenado todos os arquivos e bibliotecas necessários para um determinado projeto e funciona de modo que o interpretador, bibliotecas e *scripts* Python instalados nele fiquem isolados de outros ambientes virtuais e das demais bibliotecas instaladas no sistema operacional.

4.1.10 Diretório “manage.py”

O arquivo *manage.py* é automaticamente criado em qualquer projeto Django, de acordo com a documentação do *framework* Django disponível no site docs.djangoproject.com na seção *django-admin and manage.py*, c2005-2021. Nele fica contido várias das funções administrativas do projeto como a *runserver*, *migrate* e *makemigrations*.

4.1.11 Função “runserver”

Runserver é a função que é chamada para executar o *webserver* do projeto. Por padrão ele é iniciado na porta 8000 do endereço IP 127.0.0.1 e pode ser acessado pelo navegador da máquina local pelo endereço http://127.0.0.1:8000/.

4.1.12 Função “makemigrations”

A função *makemigrations*, apresentada no Quadro 5, cria migrações baseadas nas alterações realizadas nos *models* do projeto, ou seja, sempre que realizada alguma alteração nos *models* do projeto deve ser chamada a função *makemigrations* para criar a migração no arquivo */hosts/migrations.py* e depois realiza-se a chamada ad função *migrate* para aplicar as alterações no banco de dados.

Quadro 5 – Código-fonte Python: makemigrations.py

(continua)

```
import os
import sys

from itertools import takewhile
from django.apps import apps
from django.conf import settings
from django.core.management.base import (
    BaseCommand, CommandError, no_translations,
)
from django.db import DEFAULT_DB_ALIAS, connections, router
from django.db.migrations import Migration
from django.db.migrations.autodetector import MigrationAutodetector
from django.db.migrations.loader import MigrationLoader
from django.db.migrations.questioner import (
    InteractiveMigrationQuestioner, MigrationQuestioner,
    NonInteractiveMigrationQuestioner,
)
from django.db.migrations.state import ProjectState
from django.db.migrations.utils import get_migration_name_timestamp
from django.db.migrations.writer import MigrationWriter

class Command(BaseCommand):
    help = "Creates new migration(s) for apps."

    def add_arguments(self, parser):
        parser.add_argument(
            'args', metavar='app_label', nargs='*',
            help='Specify the app label(s) to create migrations for.',
        )
        parser.add_argument(
            '--dry-run', action='store_true',
            help="Just show what migrations would be made; don't actually
write them.",
        )
        parser.add_argument(
            '--merge', action='store_true',
            help="Enable fixing of migration conflicts.",
        )
        parser.add_argument(
            '--empty', action='store_true',
            help="Create an empty migration.",
        )
```

Quadro 5 – Código-fonte Python: makemigrations.py

(continua)

```
parser.add_argument(
            '--noinput', '--no-input', action='store_false',
dest='interactive', help='Tells Django to NOT prompt the user for input of
any kind.',
        )
        parser.add_argument(
            '-n', '--name',
            help="Use this name for migration file(s).",
        )
        parser.add_argument(
            '--no-header', action='store_false', dest='include_header',
            help='Do not add header comments to new migration file(s).',
        )
        parser.add_argument(
            '--check', action='store_true', dest='check_changes',
            help='Exit with a non-zero status if model changes are missing
migrations.',
        )

    @no_translations
    def handle(self, *app_labels, **options):
        self.verbosity = options['verbosity']
        self.interactive = options['interactive']
        self.dry_run = options['dry_run']
        self.merge = options['merge']
        self.empty = options['empty']
        self.migration_name = options['name']
        if self.migration_name and not self.migration_name.isidentifier():
            raise CommandError('The migration name must be a valid Python
identifier.')
        self.include_header = options['include_header']
        check_changes = options['check_changes']

        # Make sure the app they asked for exists
        app_labels = set(app_labels)
        has_bad_labels = False
        for app_label in app_labels:
            try:
                apps.get_app_config(app_label)
            except LookupError as err:
                self.stderr.write(str(err))
                has_bad_labels = True
        if has_bad_labels:
            sys.exit(2)

        # Load the current graph state. Pass in None for the connection so
        # the loader doesn't try to resolve replaced migrations from DB.
        loader = MigrationLoader(None, ignore_no_migrations=True)

        # Raise an error if any migrations are applied before their
dependencies.
        consistency_check_labels = {config.label for config in
apps.get_app_configs()}
        # Non-default databases are only checked if database routers used.
        aliases_to_check = connections if settings.DATABASE_ROUTERS else
[DEFAULT_DB_ALIAS]
```

Quadro 5 – Código-fonte Python: makemigrations.py

(continua)

```
        for alias in sorted(aliases_to_check):
            connection = connections[alias]
            if (connection.settings_dict['ENGINE'] !=
'django.db.backends.dummy' and any(
                    # At least one model must be migrated to the database.
                    router.allow_migrate(connection.alias, app_label,
model_name=model._meta.object_name)
                    for app_label in consistency_check_labels
                    for model in
apps.get_app_config(app_label).get_models()
            )):
                loader.check_consistent_history(connection)
# Before anything else, see if there's conflicting apps and drop out
# hard if there are any and they don't want to merge
        conflicts = loader.detect_conflicts()
# If app_labels is specified, filter out conflicting migrations for
unspecified apps
        if app_labels:
            conflicts = {
                app_label: conflict for app_label, conflict in
conflicts.items()
                if app_label in app_labels
            }
        if conflicts and not self.merge:
            name_str = "; ".join(
                "%s in %s" % (", ".join(names), app)
                for app, names in conflicts.items()
            )
            raise CommandError(
                "Conflicting migrations detected; multiple leaf nodes in
the "
                "migration graph: (%s).\nTo fix them run "
                "'python manage.py makemigrations --merge'" % name_str
            )
# If they want to merge and there's nothing to merge, then politely exit
        if self.merge and not conflicts:
            self.stdout.write("No conflicts detected to merge.")
            return
# If they want to merge and there is something to merge, then
# divert into the merge code
        if self.merge and conflicts:
            return self.handle_merge(loader, conflicts)
        if self.interactive:
            questioner =
InteractiveMigrationQuestioner(specified_apps=app_labels,
dry_run=self.dry_run)
        else:
            questioner =
NonInteractiveMigrationQuestioner(specified_apps=app_labels,
dry_run=self.dry_run)
        # Set up autodetector
        autodetector = MigrationAutodetector(
            loader.project_state(),
            ProjectState.from_apps(apps),
            questioner,
        )
```

Quadro 5 – Código-fonte Python: makemigrations.py

(continua)

```
# If they want to make an empty migration, make one for each app
        if self.empty:
            if not app_labels:
                raise CommandError("You must supply at least one app label
when using --empty.")
            # Make a fake changes() result we can pass to
arrange_for_graph
            changes = {
                app: [Migration("custom", app)]
                for app in app_labels
            }
            changes = autodetector.arrange_for_graph(
                changes=changes,
                graph=loader.graph,
                migration_name=self.migration_name,
            )
            self.write_migration_files(changes)
            return

        # Detect changes
        changes = autodetector.changes(
            graph=loader.graph,
            trim_to_apps=app_labels or None,
            convert_apps=app_labels or None,
            migration_name=self.migration_name,
        )

        if not changes:
            # No changes? Tell them.
            if self.verbosity >= 1:
                if app_labels:
                    if len(app_labels) == 1:
                        self.stdout.write("No changes detected in app
'%s'" % app_labels.pop())
                    else:
                        self.stdout.write("No changes detected in apps
'%s'" % ("', '".join(app_labels)))
                else:
                    self.stdout.write("No changes detected")
        else:
            self.write_migration_files(changes)
            if check_changes:
                sys.exit(1)

    def write_migration_files(self, changes):
        """
        Take a changes dict and write them out as migration files.
        """
        directory_created = {}
        for app_label, app_migrations in changes.items():
            if self.verbosity >= 1:
                self.stdout.write(self.style.MIGRATE_HEADING("Migrations
for '%s':" % app_label))
```

Quadro 5 – Código-fonte Python: makemigrations.py

(continua)

```
            for migration in app_migrations:
                # Describe the migration
                writer = MigrationWriter(migration, self.include_header)
                if self.verbosity >= 1:
# Display a relative path if it's below the current working
# directory, or an absolute path otherwise.
                    try:
                        migration_string = os.path.relpath(writer.path)
                    except ValueError:
                        migration_string = writer.path
                    if migration_string.startswith('..'):
                        migration_string = writer.path
                    self.stdout.write('  %s\n' %
self.style.MIGRATE_LABEL(migration_string))
                    for operation in migration.operations:
                        self.stdout.write('    - %s' %
operation.describe())
                if not self.dry_run:
                    # Write the migrations file to the disk.
                    migrations_directory = os.path.dirname(writer.path)
                    if not directory_created.get(app_label):
                        os.makedirs(migrations_directory, exist_ok=True)
                        init_path = os.path.join(migrations_directory,
"__init__.py")
                        if not os.path.isfile(init_path):
                            open(init_path, "w").close()
                        # We just do this once per app
                        directory_created[app_label] = True
                    migration_string = writer.as_string()
                    with open(writer.path, "w", encoding='utf-8') as fh:
                        fh.write(migration_string)
                elif self.verbosity == 3:
# Alternatively, makemigrations --dry-run --verbosity 3
# will output the migrations to stdout rather than saving
# the file to the disk.
                    self.stdout.write(self.style.MIGRATE_HEADING(
                        "Full migrations file '%s':" % writer.filename
                    ))
                    self.stdout.write(writer.as_string())
    def handle_merge(self, loader, conflicts):
        """
        Handles merging together conflicted migrations interactively,
        if it's safe; otherwise, advises on how to fix it.
        """
        if self.interactive:
            questioner = InteractiveMigrationQuestioner()
        else:
            questioner = MigrationQuestioner(defaults={'ask_merge': True})
        for app_label, migration_names in conflicts.items():
# Grab out the migrations in question, and work out their
# common ancestor.
            merge_migrations = []
            for migration_name in migration_names:
              migration = loader.get_migration(app_label, migration_name)
              migration.ancestry = [
    mig for mig in loader.graph.forwards_plan((app_label, migration_name))
                    if mig[0] == migration.app_label
                ]
                merge_migrations.append(migration)
```

Quadro 5 – Código-fonte Python: makemigrations.py

(continua)

```
            def all_items_equal(seq):
                return all(item == seq[0] for item in seq[1:])
            merge_migrations_generations = zip(*(m.ancestry for m in
merge_migrations))
            common_ancestor_count = sum(1 for common_ancestor_generation
                                        in takewhile(all_items_equal,
merge_migrations_generations))
            if not common_ancestor_count:
                raise ValueError("Could not find common ancestor of %s" %
migration_names)
            # Now work out the operations along each divergent branch
            for migration in merge_migrations:
                migration.branch =
migration.ancestry[common_ancestor_count:]
                migrations_ops = (loader.get_migration(node_app,
node_name).operations
                                  for node_app, node_name in
migration.branch)
                migration.merged_operations = sum(migrations_ops, [])
            # In future, this could use some of the Optimizer code
            # (can_optimize_through) to automatically see if they're
            # mergeable. For now, we always just prompt the user.
            if self.verbosity > 0:
                self.stdout.write(self.style.MIGRATE_HEADING("Merging %s"
% app_label))
                for migration in merge_migrations:
                    self.stdout.write(self.style.MIGRATE_LABEL("  Branch
%s" % migration.name))
                    for operation in migration.merged_operations:
                        self.stdout.write('    - %s' %
operation.describe())
            if questioner.ask_merge(app_label):
                # If they still want to merge it, then write out an empty
                # file depending on the migrations needing merging.
                numbers = [
                    MigrationAutodetector.parse_number(migration.name)
                    for migration in merge_migrations
                ]
                try:
                    biggest_number = max(x for x in numbers if x is not
None)
                except ValueError:
                    biggest_number = 1
                subclass = type("Migration", (Migration,), {
                    "dependencies": [(app_label, migration.name) for
migration in merge_migrations],
                })
                migration_name = "%04i_%s" % (
                    biggest_number + 1,
                    self.migration_name or ("merge_%s" %
get_migration_name_timestamp())
                )
                new_migration = subclass(migration_name, app_label)
                writer = MigrationWriter(new_migration,
self.include_header)
                if not self.dry_run:
                    # Write the merge migrations file to the disk
                    with open(writer.path, "w", encoding='utf-8') as fh:
                        fh.write(writer.as_string())
```

Quadro 5 – Código-fonte Python: makemigrations.py

(conclusão)

```
                    if self.verbosity > 0:
                        self.stdout.write("\nCreated new merge migration
%s" % writer.path)
                elif self.verbosity == 3:
                    # Alternatively, makemigrations --merge --dry-run --
verbosity 3
                    # will output the merge migrations to stdout rather
than saving
                    # the file to the disk.
                    self.stdout.write(self.style.MIGRATE_HEADING(
                        "Full merge migrations file '%s':" %
writer.filename
                    ))
                    self.stdout.write(writer.as_string())
```

Fonte: Autoria própria (2022).

4.1.13 Função “migrate”

A função *migrate* é utilizada para sincronizar as migrações do diretório */hosts/migrations* e aplicar ao banco de dados configurado. As migrações são realizadas com base no arquivo *models.py* no diretório da aplicação do projeto e quando chamada a função *migrate* deve ser passado como argumento qual o projeto se refere a migração.

4.2 Banco de dados

O banco de dados utilizado pelo projeto é o SQLite3 e o nome do arquivo dentro do projeto é o db.sqlite3 e na Tabela 1, é possível visualizar todas as tabelas do banco de dados que são utilizadas nesse projeto e sua respectiva finalidade.

Tabela 1 – Tabelas do banco de dados do projeto

(continua)

Ordem	Identificação da Tabela	Finalidade
1	auth_group	Armazena os grupos de usuários criados no módulo Django-admin que auxilia o desenvolvedor na manutenção do banco de dados.
2	auth_group_permissions	Relaciona os registros das tabelas auth_group e auth_permission.
3	auth_permissions	Armazena as permissões criadas no módulo Django-admin que auxilia o desenvolvedor na manutenção do banco de dados.
4	auth_user	Armazena os usuários criados no módulo Django-admin que auxilia o desenvolvedor na manutenção do banco de dados.
5	auth_user_groups	Relaciona os registros das tabelas auth_user e auth_group.

Tabela 1 – Tabelas do banco de dados do projeto

(conclusão)

Ordem	Identificação da Tabela	Finalidade
6	auth_user_user_permissions	Relaciona os registros das tabelas auth_user e auth_permissions.
7	django_admin_log	Armazena as alterações realizadas pelos usuários no banco de dados através do módulo Django-Admin.
8	django_celery_beat_clockedscheduler	Armazena os horários agendados de execução das tasks criadas pela aplicação (Não está sendo utilizada nesse projeto).
9	django_celery_beat_crontabscheduler	Armazena as informações de agendamento de tasks criadas pela aplicação (Não está sendo utilizada nesse projeto).
10	django_celery_beat_intervalscheduler	Armazena os intervalos de execução das tasks criadas pela aplicação.
11	django_celery_beat_periodictask	Armazena as tasks que serão executadas na aplicação.
12	django_celery_beat_solarscheduler	Armazena as regras de execução de tasks que devem ser executadas de acordo com o nascer ou por do sol informando nessa tabela a latitude e longitude do local (Não está sendo utilizada nesse projeto).
13	django_celery_beat_results_taskresult	Armazena o resultado das tasks executadas.
14	django_content_type	Armazena uma lista de todas as tabelas padrões do framework Django com seu respectivo tipo.
15	django_migrations	Armazena as migrações realizadas no banco de dados.
16	django_session	Armazena as sessões dos usuários que logam no módulo Django-admin para auxílio aos desenvolvedores.
17	hosts_host	Armazena os equipamentos cadastrados pelo usuário que devem ser monitorados na rede.
18	hosts_host_host_templete	Relaciona o as tabelas hosts_host e hosts_template.
19	hosts_item	Armazena as informações que devem ser monitoradas nos equipamentos cadastrados.
20	hosts_template	Armazenas os templates cadastrados pelo usuário.
21	hosts_template_template_item	Relaciona as tabelas hosts_template e hosts_item
22	snmp_get_DD_MM_AA_hh_mm_ss	Essas tabelas são criadas e deletadas dinamicamente no código tendo como seu nome a data e horário exato de sua criação. Elas servem para armazenar as informações coletadas no equipamento cadastrado que está sendo monitorado.
23	sqlite_mater	Essa é uma tabela de manutenção do banco de dados que armazena todas as tabelas criadas e suas respectivas características.

Fonte: Autoria própria (2022).

4.2.1 Tabela de dados: django_celery_beat_intervalschedule

A tabela “django_celery_beat_intervalschedule”, apresentada na Tabela 2, armazena intervalos que serão feitos os *snmp-requests* aos dispositivos cadastrados. Sempre que realizado um cadastro de um item na interface do usuário onde é informado a periodicidade com que quer que seja executado o *snmp-request,* o sistema executa um algoritmo que verifica se o intervalo cadastrado já existe na tabela e caso não exista ele insere na tabela com um novo intervalo.

Tabela 2 – Tabela de dados: django_celery_beat_intervalschedule

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	Id	Número (Inteiro)	Identificar os registros da tabela
2	Every	Número (Inteiro)	O valor numérico do intervalo que será executado as tasks
3	Period	Varchar (24)	Unidade de medida que será utilizada para executar as tasks como: segundos, minutos e horas.

Fonte: Autoria própria (2022).

4.2.2 Tabela de dados: django_celery_beat_periodictask

A tabela “django_celery_beat_periodictask”, apresentada na Tabela 3, armazena as tarefas periódicas que deverão ser executadas no intervalo que estiver apontando na tabela *django_celery_beat_intervalschedule*. As tarefas periódicas dessa tabela são classificadas em dois tipos e são indicadas pela descrição na coluna *name*. São elas:

- SNMPGETTASK: Essa tarefa tem como objetivo realizar um snmp-request no dispositivo apontado. Entre os vários parâmetros que esse registro deve receber, os principais são a tabela em que deve ser feito a inserção dos resultados, o intervalo em que deve ser executado e a função que deve ser executada.
- CLEANDATATASK: Essa tarefa tem como objetivo realizar a exclusão dos resultados dos *snmp-requests* das tabelas depois do tempo cadastrado na tela de cadastro de itens do usuário. Essa tarefa foi desenvolvida para realizar a exclusão de registros antigos conforme o tempo determinado pelo usuário. Assim como o *SNMPGETTASK* os principais parâmetros passados a essa tarefa são a tabela em que deve ser feito a exclusão dos resultados, a função que deve ser executada e qual o tempo que deve ser considerado para realizar a exclusão dos resultados.

Tabela 3 – Tabela de dados: django_celery_beat_periodictask

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	Id	Número (Inteiro)	Identificar os registros da tabela.
2	Name	Varchar (200)	Nome da task.
3	Task	Varchar (200)	Caminho da função que será executada.
4	Args	Texto	Argumentos da função que será executada.
5	Kwargs	Texto	Campo para caso a função receba kwargs. (Opcional)
6	Queue	Varchar (200)	Nome da fila do *message broker* que for organizar a fila de tasks. (Opcional)
7	Exchange	Varchar (200)	Argumento de *exchange* para o *message broker* que for organizar a fila de tasks. (Opcional)
8	routing_key	Varchar (200)	Argumento de *routing_key* para o *message broker* que for organizar a fila de tasks. (Opcional)
9	Expires	Data	Data para expiração da task. (Opcional)
10	Enabled	Booleano	Definir a task como ativa ou inativa.
11	last_run_at	Data	Data e horário da última vez que foi executada.
12	total_run_count	Número (Inteiro sem sinal)	Total de vezes que foi executada.
13	date_changed	Data	Data da última alteração.
14	Description	Texto	Descrição da task. (Opcional)
15	crontab_id	Número (Inteiro)	Id do registro da tabela django_celery_beat_crontabscheduler.
16	interval_id	Número (Inteiro)	Id do registro da tabela django_celery_beat_intervalscheduler.
17	solar_id	Número (Inteiro)	Id do registro da tabela django_celery_beat_solarscheduler.
18	one_off	Booleano	Sinaliza para que a task seja inativada quando for executada.
19	start_time	Data	Data/horário agendada que é para começar a ser executada a task.
20	Priority	Número (Inteiro sem sinal)	Identificador de prioridade na ordem de execução se utilizado.
21	Headers	Texto	Cabeçalho para task. (Opcional)
22	clocked_id	Número (Inteiro)	Id do registro da tabela django_celery_beat_clockedschedule.
23	expire_seconds	Número (Inteiro sem sinal)	Tempo de expiração da task caso não tenha retorno.

Fonte: Autoria própria (2022).

4.2.3 Tabela de dados: hosts_host

A tabela “hosts_host”, apresentada na Tabela 4, é responsável por armazenar os *hosts* cadastrados pelo usuário no sistema. Os *hosts* são os dispositivos que se quer monitorar e capturar informações. As principais informações que devem ser preenchidas quando se cadastra um *host* são nome, IP, porta e a *community*.

Ao cadastrar um *host* o sistema automaticamente cria uma tabela para ele com o objetivo de armazenar os resultados dos *snmp-requests* realizados nele. O algoritmo utilizado para nomear as tabelas utiliza o prefixo snmp_get_ seguidos das datas no formato DD_MM_AAAA_hh_mm_ss onde: DD = dia; MM = mês; AAAA = ano; hh = hora; mm = minuto; ss = segundo.

Tabela 4 – Tabela de dados: hosts_host

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	id	Número (Inteiro)	Identificar o registro na tabela.
2	host_name	Varchar (100)	Nome do equipamento que será monitorado.
3	host_ip	Char (15)	Endereço IP do equipamento que será monitorado.
4	host_porta	Número (Inteiro sem sinal)	Porta do endereço IP por qual o equipamento será monitorado.
5	host_observacoes	Texto	Observações do equipamento que será monitorado.
6	host_status	Booleano	Status de ativo/inativo do equipamento que será monitorado.
7	host_nomeTabela_snmpGet	Varchar (100)	Nome da tabela onde serão armazenados os dados coletados do equipamento que será monitorado.
8	host_community	Varchar (32)	Community configurada no equipamento que será monitorado.

Fonte: Autoria própria (2022).

4.2.4 Tabela de dados: hosts_template”

Já a tabela “hosts_template”, apresentada na Tabela 5, armazena os *templates* criados pelo usuário.

Os *templates* são conjuntos de *itens* que por sua vez são as informações que se quer capturar de um *host*. O vínculo entre *hosts* e *itens* é feito através dos *templates* com o objetivo de facilitar a configurações das informações que se quer capturar do dispositivo de rede, uma vez que vinculado um *template* a um *host* é vinculado de forma automática todos os *itens* cadastrados no *template*.

Tabela 5 – Tabela de dados: hosts_template

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	Id	Número (Inteiro)	Identificar o registro na tabela.
2	template_nome	Varchar (100)	Nome do templete.
3	template_observacoes	Texto	Observações do template.

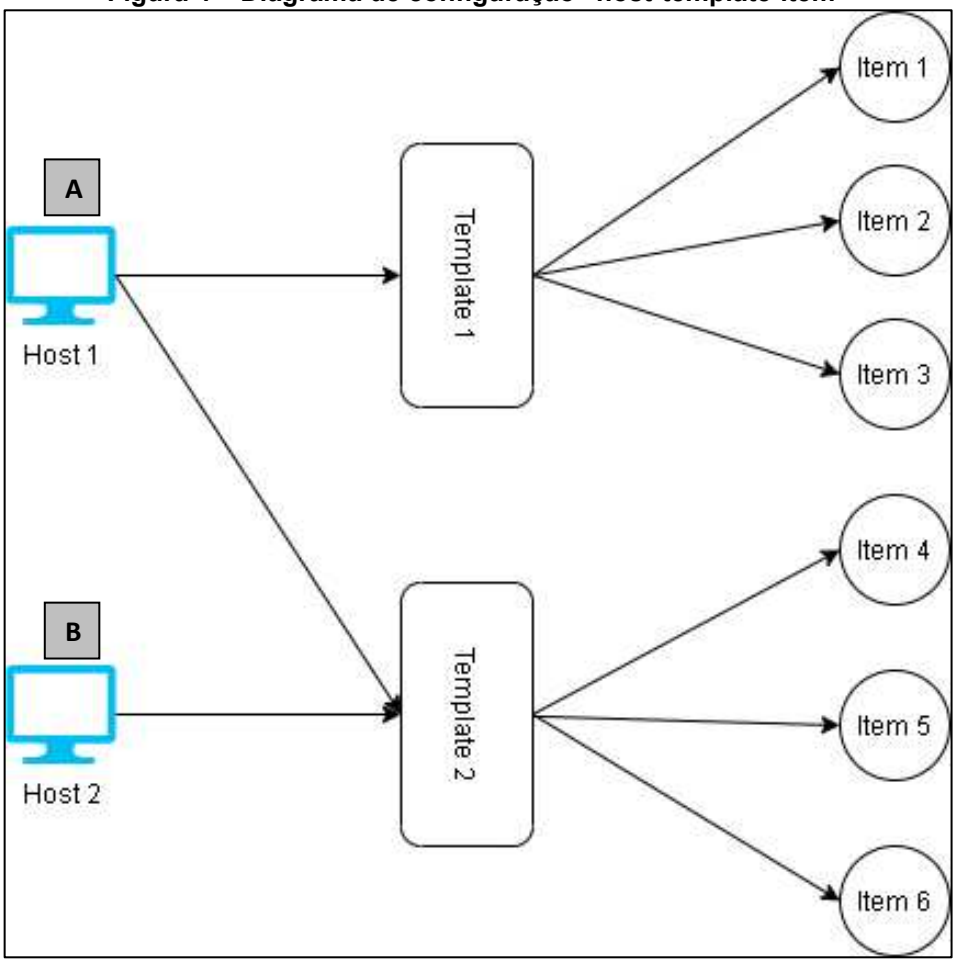
Fonte: Autoria própria (2022).

Na Figura 4 é possível observar um diagrama que exemplifica o os cadastros feitos pelo usuário e a forma que o sistema os vincula.

O *host* 1, item A na Figura 4, é um dispositivo de rede que está atrelado aos *templates* 1 e 2. Então o *host* 1 está trelado ao *item* 1,2 e 3 através do *template* 1 e 4,5 e 6 através do *template* 2.

Já o *host* 2, item B na Figura 4, está atrelado somente aos *itens* 4,5 e 6 através do *template* 2.

Figura 4 – Diagrama de configuração "host-template-item"



Fonte: Autoria própria (2022).

4.2.5 Tabela de dados: hosts_item

A tabela "hosts_item", apresentada na Tabela 6, armazena os *itens* cadastrados pelo usuário. O *item* é a informação que se quer capturar do dispositivo de rede. As principais informações que devem ser preenchidas ao cadastrar um *item* são OID, intervalo de atualização, tempo de armazenamento e tipo da informação coletada.

O tipo da informação coletada irá influenciar em como a informação será exibida na tela de *dashboard* da aplicação. São as opções:

- Carácter: exibida no *dashboard* através de um único campo de texto que se atualiza e sobrescreve o anterior;
- Numérico inteiro: exibida através de um gráfico organizado por data e horário;
- Numérico decimal: também exibida através de um gráfico organizado por data e horário;
- *Log*: exibida através de um campo de texto que vai acrescentando na linha de baixo o resultado.

Tabela 6 – Tabela de dados: hosts_item

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	id	Número (Inteiro)	Identificar os registros da tabela.
2	item_nome	Varchar (100)	Nome da informação que será monitorada no equipamento.
3	item_oid	Varchar (100)	Endereço O.I.D da MIB onde será coletada a informação que será monitorada.
4	item_tipoInformacao	Varchar (100)	O tipo de informação que se espera receber do equipamento que será monitorado como por exemplo número inteiro, número decimal, texto etc.
5	item_intervaloAtualizacao	Número (Inteiro)	Intervalo de tempo em que será executado a consulta da informação no equipamento que será monitorado.
6	item_intervaloAtualizacaoUn	Varchar (100)	Unidade de medida do intervalo de tempo em que será executado a consulta da informação no equipamento que será monitorado como por exemplo: segundos, minutos, horas.
7	item_tempoArmazenamentoDados	Número (Inteiro)	Tempo que deverá ficar armazenado os dados coletados do equipamento que será monitorado.
8	item_tempoArmazenamentoDadosUn	Varchar (100)	Unidade de medida do tempo que deverá ficar armazenado os dados coletados do equipamento que será monitorado como por exemplo: minutos, horas, dias.
9	item_expresaoConversao	Varchar (100)	Código Javascript que pode ser informado para converter o valor recebido do equipamento que será monitorado.

Fonte: Autoria própria (2022).

4.2.6 Tabela de dados: sqlite_master

A tabela “sqlite_master”, apresentada na Tabela 7, tem a função de armazenar o nome de todas as tabelas criadas no banco de dados e é utilizada para controle interno do banco de dados.

Tabela 7 – Tabela de dados: sqlite_master

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	type	Texto	Tipo de objeto no banco de dados que pode ser *table*, *index*, *view* etc.
2	name	Texto	Nome do objeto no banco de dados.
3	tbl_name	Texto	Nome da tabela associada ao objeto no banco de dados.
4	rootpage	Número (Inteiro)	A coluna rootpage armazena o número da página raiz da b-tree para tabelas e índices. Para linhas que definem exibições, gatilhos e tabelas virtuais, a coluna rootpage é 0 ou *NULL*.
5	sql	Texto	Código SQL para a criação da tabela.

Fonte: Autoria própria (2022).

4.2.7 Tabelas relacionais

Além das tabelas descritas acima também foram criadas com a ajuda da biblioteca *models* do *framewrok* Django tabelas relacionais que relacionam as tabelas de *hosts*, *templates* e *itens*, são elas:

- *hosts_host_host_template*: O objetivo dela é relacionar o *host* ao *template*. Ela recebe o número de identificação (ID) do *host* da tabela *hosts_host* e o ID do *template* da tabela *hosts_template*. Ela é preenchida quando vinculado os *templates* ao *host* no cadastro de *hosts* através do campo *ManyToManyField* que significa que um *host* pode estar relacionado a vários *templates* assim como um *template* pode estar relacionado a vários *hosts*;
- *hosts_template_template_item*: O objetivo dela é *relacionar* o *template* ao *item*. Ela recebe o ID do *template* da tabela *hosts_template* e o ID do *item* da tabela *hosts_item*. Ela é preenchida quando vinculado um *item* ao um *template* na tela de cadastro de *templates* através do campo *ManyToManyField* que significa que um *template* pode estar relacionado a vários *itens* assim como um *item* pode estar relacionado a vários *templates*.

Na Tabela 8, tem-se uma explicação da finalidade de todos os campos da tabela “hosts_template_template_item”.

Tabela 8 – Tabela de dados: hosts_template_template_item

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	id	Número (Inteiro)	Identificação dos registros na tabela.
2	template id	Número (Inteiro)	Id do template da tabela hosts_template.
3	item_id	Número (Inteiro)	Id do item da tabela hosts_item.

Fonte: Autoria própria (2022).

Na Tabela 9, estão os campos e suas respetivas finalidades referente a tabela “hosts_host_host_template”.

Tabela 9 – Tabela de dados: hosts_host_host_template

Ordem	Nome do Campo	Tipo do Dado	Finalidade
1	Id	Número (Inteiro)	Identificação dos registros na tabela.
2	host_id	Número (Inteiro)	Id do host da tabela hosts_host
3	template_id	Número (Inteiro)	Id do template da tabela hosts_template.

Fonte: Autoria própria (2022).

4.3 Front-end

O *front-end* foi desenvolvido com o *framework* Angular e a biblioteca adicional de *user interface* PrimeNG. O projeto também foi desenvolvido utilizando o conceito de *Single Page Application* (SPA), ou seja, todo o projeto tem apenas uma página principal e suas funcionalidades são módulos dessa página. Tudo que o usuário vê na página são módulos e estes módulos são carregados apenas quando chamados. Abaixo temos a principais pastas e arquivos do projeto que ficam dentro da pasta app.

4.3.1 Arquivos página inicial

Na página inicial temos 3 (três) arquivos: a) app.component.ts; b) app.component.css; e c) app.component.html. Eles compõem a página principal que é carregada ao acessar o *front-end*. A partir dessa página todos os outros módulos são carregados de acordo com a necessidade do usuário. Esses 3 arquivos em questão são os componentes base do aplicativo de onde o framework é inicializado.

O framework Angular é todo baseado em componentes. Os componentes são códigos que podem ser reutilizados no projeto e realizam as mais diversas tarefas, como por exemplo login ou mostrar uma tabela. Todos os componentes são compostos pelo arquivo HTML que declara tudo que é renderizado em uma página, um seletor CSS que define como o componente é usado em um modelo e um arquivo Typescript que define o comportamento.

4.3.2 Página inicial home-tab

A página inicial Home-tab, apresentada na Figura 5, é principal página que o usuário acessa, contém um menu de navegação para as outras tabs e também informações básicas como os números de *hosts*, *templates* e itens.

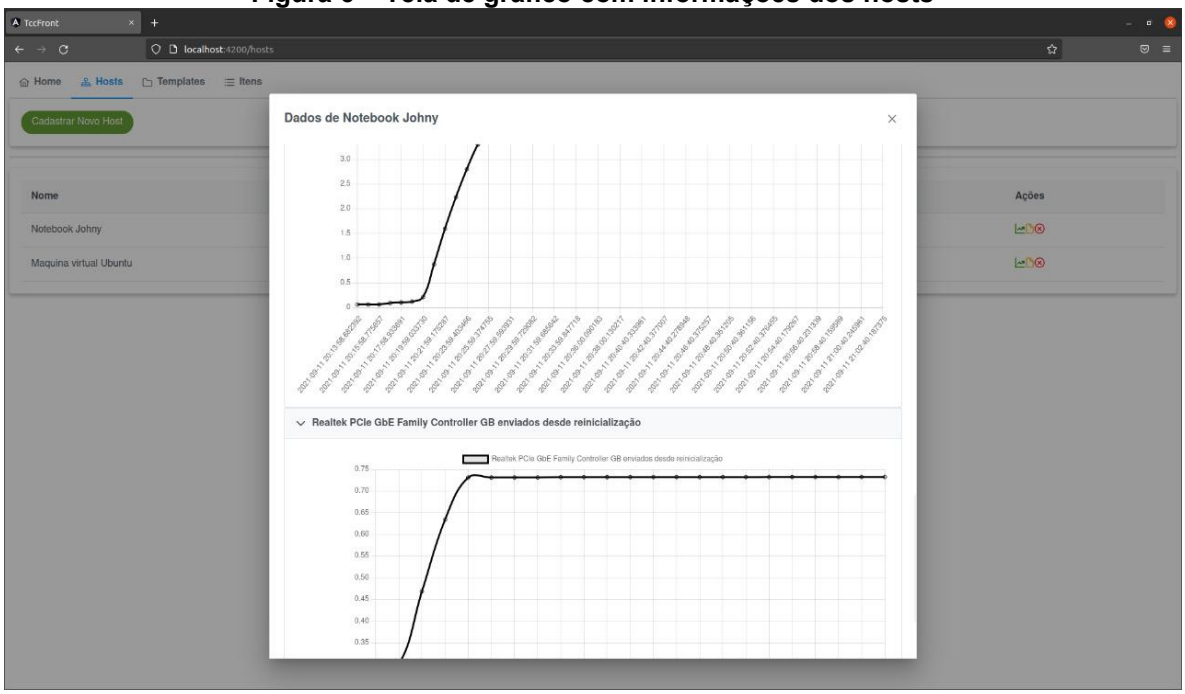
Figura 5 – Tela da página inicial da aplicação

Fonte: Autoria própria (2022).

4.3.3 Página chart-detail

A página Chart-detail, apresentada na Figura 6, é responsável por apresentar as informações coletadas em forma de gráfico, pode ser acessada através da tabela com os *hosts*.

Figura 6 – Tela de gráfico com informações dos hosts

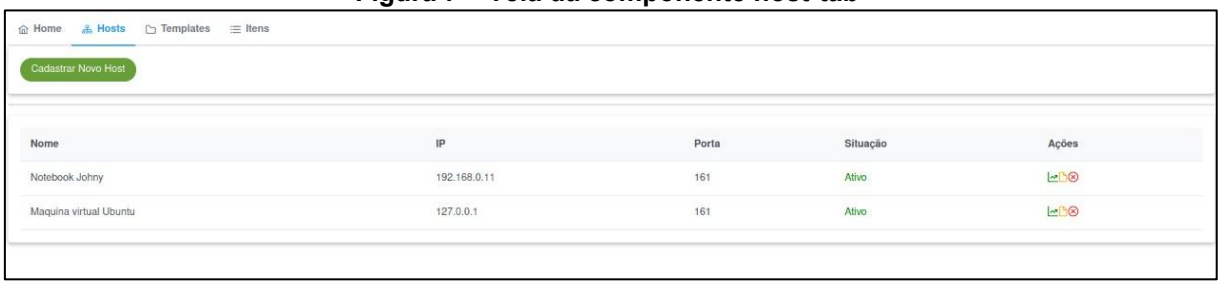


Fonte: Autoria própria (2022).

4.3.4 Componente hosts-tab

A hosts-tab, apresentado na Figura 7, é um componente que contém uma tabela com todos os *hosts* cadastrados, através dela é possível adicionar um novo *host*, editar ou apagar um *host* existente e acessar os gráficos referentes a um host específico.

Figura 7 – Tela da componente host-tab



Fonte: Autoria própria (2022).

4.3.5 Componente host-detail

O componente host-detail, apresentado na Figura 8, é um formulário para poder criar ou editar um *host*, no caso de uma edição as informações já aparecem preenchidas e podem ser editadas.

Figura 8 – Tela de edição/criação de host

Fonte: Autoria própria (2022).

4.3.6 Componente template-tab

A componente template-tab, apresentado na Figura 9, contém uma tabela que lista todos os *templates* cadastrados, nela é possível cadastrar novos *templates* ou editar e/ou apagar *templates* existentes.

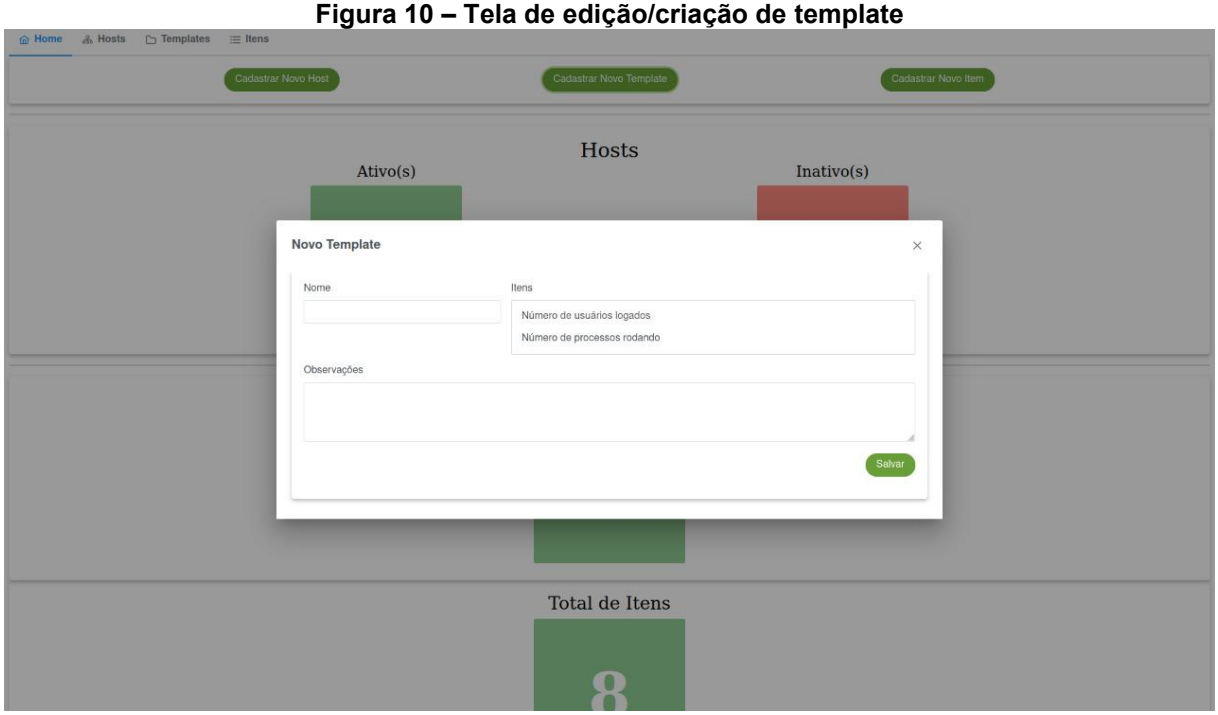
Figura 9 – Tela da componente template-tab

Fonte: Autoria própria (2022).

4.3.7 Componente template-detail

O componente template-detail, apresentado na Figura 10, é um formulário para poder criar ou editar um *template*, no caso de uma edição as informações já vêm preenchidas e podem ser editadas.

Figura 10 – Tela de edição/criação de template

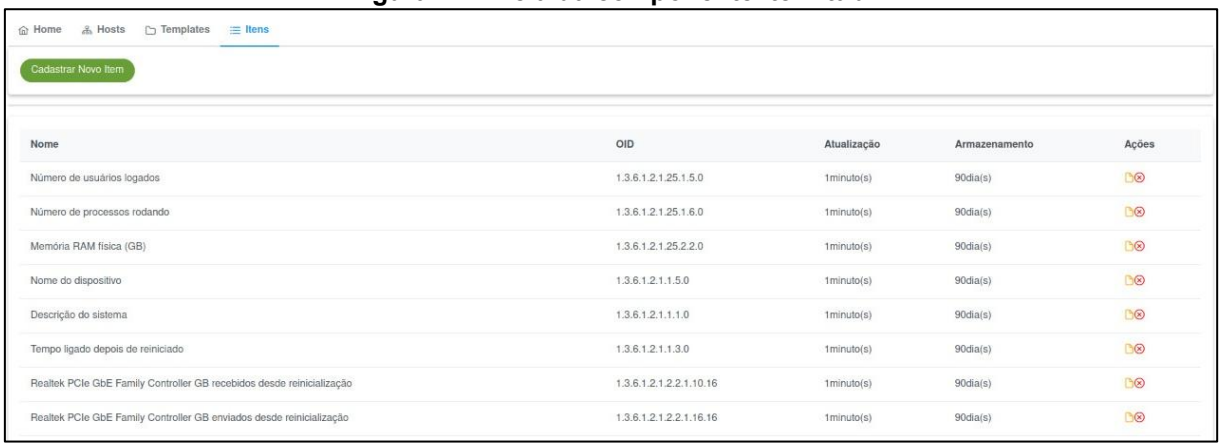


Fonte: Autoria própria (2022).

4.3.8 Componente item-tab

O componente item-tab, apresentado na Figura 11, contém uma tabela que lista todos os itens cadastrados que podem ser usados para solicitar informação pelo protocolo SNMP, nela é possível cadastrar novos itens ou editar e/ou apagar itens existentes.

Figura 11 – Tela da componente item-tab

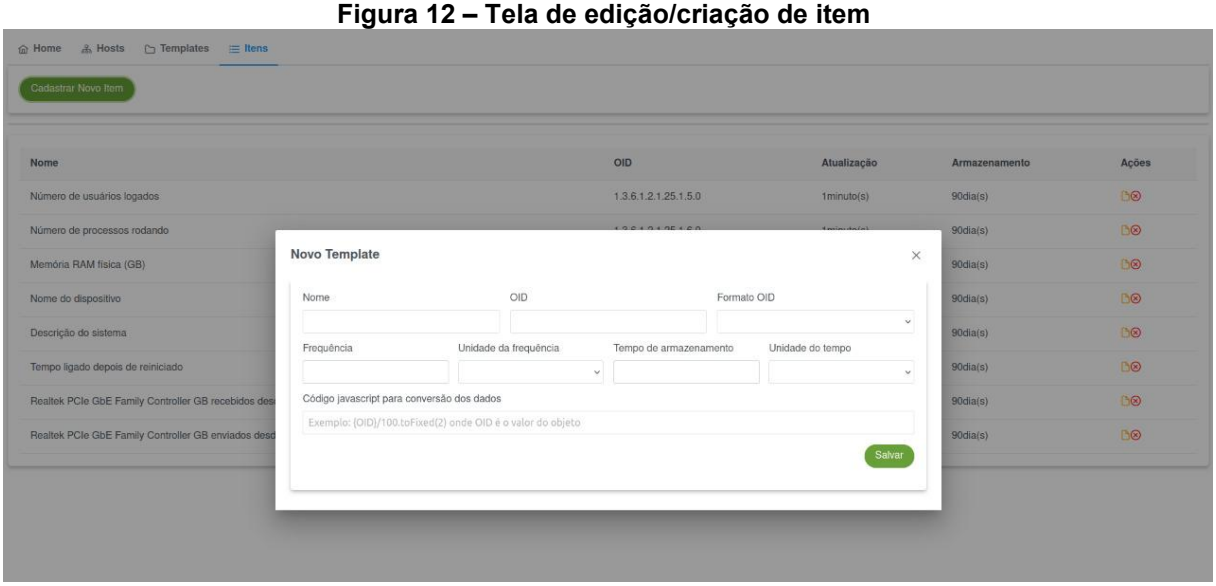


Fonte: Autoria própria (2022).

4.3.9 Componente item-detail

A componente item-detail, apresentado na Figura 12, é um formulário para poder criar ou editar um item, no caso de uma edição as informações já vêm preenchidas e podem ser editadas.

Figura 12 – Tela de edição/criação de item



Fonte: Autoria própria (2022).

4.3.10 Arquivo models

O Arquivo *models* contém os modelos dos objetos que são recebidos do *back-end*. Através deles é possível fazer a desserialização/serialização desses objetos.

4.3.11 Elemento de comunicação service

O elemento *service* é responsável pela comunicação efetiva com o *back-end*. Ele que realiza as requisições *rest* para busca e persistência dos dados que são exibidos na tela.

5 CONSIDERAÇÕES FINAIS

Durante o desenvolvimento desse trabalho pudemos observar e aplicar diversos conhecimentos adquiridos durante o Curso Superior de Tecnologia em Sistemas de Telecomunicação. As diversas disciplinas cursadas foram extremamente úteis para a elaboração da ferramenta de monitoramento de rede.

Apesar das limitações impostas pelo período de pandemia, esse trabalho pôde chegar bem próximo da idéia inicial.

A utilização da linguagem de programação Python foi de grande valia pois, apesar de não termos uma disciplina específica dessa linguagem, há muito conteúdo disponível na internet e a comunidade que utiliza essa linguagem é muito ativa.

Desde a pesquisa pelas tecnologias que pretendíamos utilizar e a verificação de sua viabilidade até os problemas encontrados durante o desenvolvimento, tudo foi importante para o aprendizado final que veio de forma complementar a tudo o que aprendemos durante o curso.

O grande desafio para a criação dessa ferramenta foi cumprir todas as etapas e realizar as integrações entre as diversas tecnologias e linguagens de programação utilizada no projeto, assim como entender como elas funcionavam e como utilizar da melhor forma possível.

Por mais que se tenha chegado muito próximo do resultado esperado inicialmente, nessa ferramenta não foi implementado um sistema de *login* com senhas e acessos com credenciais para que fosse possível ter uma ferramenta acabada com todos os requisitos de segurança atendidos.

Outra limitação do projeto é o monitoramento de características de um equipamento que necessitam de vários objetos do protocolo SNMP e depois a conversão/cálculo para se obter uma informação mais clara. A exemplo disso é o uso de CPU que normalmente ficam armazenados em diferentes variáveis e necessitam de demais informações e interações para se obter uma informação mais clara da utilização. Entende-se também que isso poderia ser algo a ser implementado em trabalhos futuros.

REFERÊNCIAS

ALVARENGA, Igor Drummond; RAMOS, Bruno Lange. **Simple Network Management Protocol (SNMP).** Universidade Federal do Rio de Janeiro - Departamento de Engenharia Eletrônica - EEL878 (Redes de Computadores I), jun. 2011. Disponível em: <https://www.gta.ufrj.br/grad/11_1/snmp/index.html>. Acesso em: 20 de set. 2021.

ANGULAR. **Angular CLI**. Copyright© Google, última revisão em: 28 out. 2021. Disponível em: <https://angular.io/guide/what-is-angular#angular-cli>. Acesso em: 24 nov. 2021.

ARS_TECHNICA. **Microsoft TypeScript: o JavaScript de que precisamos ou uma solução à procura de um problema.** Copyright© Condé Nast, publicado em: 10 fev. 2012. Disponível em: <https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/>. Acesso em: 24 nov. 2021.

CARYULY, Rosales Briceño**. Protocolo SNMP (protocolo sencillo de administración de redes)**. Télématique: Revista Electrónica de Estudios Telemáticos, 2004, v. 3(1), p. 94-106.

BRUDO, Ilan. **Top 40+ VSCode Extensions for Developers in 2022**. Publicado em: 02 dez. 2021. Disponível em: <https://www.tabnine.com/blog/top-vscode-extensions/>. Acesso em: 24 dez. 2021.

DJANGO. **Django: Documentation | Applications**. Copyright© Django Software Foundation and individual contributors, publicado em: 22 ago. 2008. Disponível em: <https://docs.djangoproject.com/en/3.2/ref/applications/>. Acesso em: 11 nov. 2021.

DJANGO. **Django: Getting started with Django.** Copyright© Django Software Foundation and individual contributors, publicado em: 16 dez. 2014. Disponível em: <https://www.djangoproject.com/start/>. Acesso em: 01 abr. 2021.

DJANGO. **Django: First steps with Django | Using Celery with Django**. Copyright© Django Software Foundation and individual contributors. Celery 5.2.3 documentation, publicado em: 22 abr. 2021. Disponível em: <https://docs.celeryproject.org/en/stable/django/first-steps-with-django.html>. Acesso em: 11 nov. 2021.

ETINGOF, Ilya. **SNMP Versions**. Copyright© Ilya Etingof, publicado em: 8 set. 2020. Disponível em: <https://pysnmp.readthedocs.io/en/latest/examples/hlapi/v3arch/asyncore/sync/manager/cmdgen/snmp-versions.html>. Acesso em: 17 nov. 2021.

FALBRIARD, Claude. **Protocolos e aplicações para redes de computadores.** 1. ed. São Paulo: Érica, 2001.

GITHUB. **Microsoft: TypeScript**. Copyright© GitHub, Inc., publicado em: 17 ago. 2019. Disponível em: <https://github.com/microsoft/TypeScript >. Acesso em: 24 nov. 2021.

JAVATPOINT. **TypeScript Version**. Copyright© www.javatpoint.com, publicado em: 15 fev. 2019. Disponível em: <https://www.javatpoint.com/typescript-versions>. Acesso em: 24 nov. 2021.

JETBRAINS. **PyCharm: O IDE Python para desenvolvedores profissionais.** PyCharm Educational Edition. Copyright© JetBrains s.r.o., publicado em: 25 abr. 2020. Disponível em: <https://www.jetbrains.com/pt-br/pycharm/>. Acesso em: 23 abr. 2021.

KLEIS, Elton Gastardelli. **Redes Definidas por SW I**. Copyright© Teleco, publicado em: 11 mai. 2015. Disponível em: <https://www.teleco.com.br/tutoriais/tutorialsw1/pagina_2.asp>. Acesso em: 13 set. 2019.

MCKEOWN, Nick; *et al*. **Openflow: Enabling innovation in campus networks**. ACM SIGCOMM Computer Communication Review, v. 38, n. 2, abr. 2008. Disponível em: <http://ccr.sigcomm.org/online/files/p69-v38n2n-mckeown.pdf>. Acesso em: 28 out. 2019.

MONTE, Luis Renato; BERNARDO, Rodrigo; OLIVEIRA, Vinícius Garcia de. **Protocolos de roteamento e protocolos TCP/UDP em redes Ad-Hoc.** Copyright© Teleco, publicado em: 01 ago. 2005. Disponível em: <https://www.teleco.com.br/tutoriais/tutorialprotocolo/default.asp>. Acesso em: 20 set. 2021.

MOZILLA. **HTML: Linguagem de Marcação de Hipertexto.** Copyright© Mozilla and individual contributors. Tutoriais para iniciantes, publicado em: 20 jun. 2012. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 01 mai. 2021.

MOZILLA. **CSS.** Copyright© Mozilla and individual contributors. Tutoriais, publicado em: 16 dez. 2018. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Acesso em: 24 nov. 2021.

PYTHON. **Python: venv - Criação de ambientes virtuais**. Copyright© Python Software Foundation, publicado em: 26 abr. 2020. Disponível em: <https://docs.python.org/pt-br/3/library/venv.html>. Acesso em: 26 jul. 2021.

RABBITMQ. **RabbitMQ: RabbitMQ is the most widely deployed open source message broker.** Copyright© VMware, Inc., publicado em: 10 ago. 2011. Disponível em: <https://www.rabbitmq.com/>. Acesso em: 29 abr. 2021.

ROSENWASSER, Daniel. **TypeScript 2.0 is now available!** Copyright© Microsoft, publicado em: 22 set. 2016. Disponível em: <https://devblogs.microsoft.com/typescript/announcing-typescript-2-0/>. Acesso em: 24 nov. 2021.

ROSENWASSER, Daniel. **Announcing TypeScript 3.0**. Copyright© Microsoft, publicado em: 30 jul. 2018. Disponível em: <https://devblogs.microsoft.com/typescript/announcing-typescript-2-0/>. Acesso em: 24 nov. 2021.

SILVA, Marcela Sisiliani de Sena. **[RabbitMQ] Introdução ao mundo das filas.** Copyright© iMasters, publicado em: 28 mai, 2019. Disponível em: <https://imasters.com.br/back-end/rabbitmq-introducao-ao-mundo-das-filas>. Acesso em: 29 abr. 2021.

SOARES, Alexandre Seixas; *et al*. **Redes de Computadores I**. Universidade Federal do Rio de Janeiro (UFRJ). Curso de Engenharia de Controle e Automação. Turma de Engenharia de Controle e Automação, 9º período. Publicado em: 21 ago. 2010. Disponível em: <https://www.gta.ufrj.br/grad/10_1/snmp/versoes.htm>. Acesso em: 11 nov. 2021.

TANENBAUM, Andrew Stuart. **Redes de Computadores**. 5. ed. São Paulo: Pearson, 2011.

TOKIO. **A história do Python. As versões de uma linguagem única.** Copyright© Tokio New Technology School, publicado em: 07 jul. 2021. Disponível em: <https://tokioschool.pt/noticias/historia-python/>. Acesso em: 23 nov. 2021.