UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ANDRÉ LUÍS MALDONADO DANIEL

AUTOMATIZAÇÃO DE CONFIGURAÇÃO DE MÚLTIPLOS SWITCHES EM UMA REDE UTILIZANDO UMA SBC COM GERÊNCIA REMOTA E SISTEMA WEB.

CAMPO MOURÃO

2022

ANDRÉ LUÍS MALDONADO DANIEL

AUTOMATIZAÇÃO DE CONFIGURAÇÃO DE MÚLTIPLOS SWITCHES EM UMA REDE UTILIZANDO UMA SBC COM GERÊNCIA REMOTA E SISTEMA WEB.

Automation of Multiple Switches Configuration in a network using an SBC with remote management and Web System.

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica do Curso de Bacharelado em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Paulo Denis Garcez da Luz

Coorientador: Prof. Dr. Roberto Wilhelm Krauss Martinez

CAMPO MOURÃO

2022

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
UTFPR - CAMPUS CAMPO MOURÃO
DIRETORIA-GERAL - CAMPUS CAMPO MOURÃO
DIRETORIA DE GRAD. EDUC. PROFISSIONAL-CM
SECRETARIA DE GESTAO ACADEMICA-CM
DEP. ACADEMICO DE ELETRONICA-CM

INFORMAÇÃO

TERMO DE APROVAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO INTITULADO:

AUTOMATIZAÇÃO DE CONFIGURAÇÃO DE MÚLTIPLOS SWITCHES EM UMA REDE UTILIZANDO UMA SBC COM GERÊNCIA REMOTA E SISTEMA WEB

DO DISCENTE: ANDRÉ LUÍS MALDONADO DANIEL

Trabalho de Conclusão de Curso apresentado no dia **07 de Novembro de 2022** ao Curso Superior de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, Campus Campo Mourão. O discente foi arguido pela Comissão Examinadora composta pelos professores abaixo assinados. Após deliberação, a comissão considerou o trabalho **aprovado com alterações**.

_____

Prof. Dr. Leandro Castilho Brolin
Avaliador 1 - UTFPR
(assinado eletronicamente)

_____

Prof. Dr. Marcelo Nanni
Avaliador 2 - UTFPR
(assinado eletronicamente)

_____

Prof. Dr. Paulo Denis Garcez da Luz
Orientador - UTFPR
(assinado eletronicamente)

Campo Mourão, 07 de novembro de 2022.

Documento assinado eletronicamente por (Document electronically signed by) **MARCELO NANNI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 07/11/2022, às 10:21, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por (Document electronically signed by) **PAULO DENIS GARCEZ DA LUZ**, **PROFESSOR(A) ORIENTADOR(A)**, em (at) 07/11/2022, às 10:21, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por (Document electronically signed by) **LEANDRO CASTILHO BROLIN**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 07/11/2022, às 10:21, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site (The authenticity of this document can be checked on the website) https://sei.utfpr.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador (informing the verification code) **3089367** e o código CRC (and the CRC code) **26614D90**.

AGRADECIMENTOS

Agradeço a minha família, por me ajudar a chegar até aqui.

A minha namorada por estar ao meu lado e me apoiar.

Meu orientador e coorientador, por me guiar durante este trabalho.

E, também ao colegas e amigos que fiz nessa etapa da minha vida.

Estas poucas linhas não mostram todo o apoio e ajuda que recebi durante essa jornada que trilhei, mas com certeza não teria chegado tão longe sem eles.

RESUMO

O presente trabalho tem como objetivo apresentar o desenvolvimento de um grupo de softwares de sistemas embarcados para a gerência remota de *switches*. O sistema foi desenvolvido visando baixo custo ao se utilizar de uma Raspberry Pi Model B +. Para o desenvolvimento do sistema foi estudado a interface web de um *switch* TL-SG108E da TP-Link. Com o estudo da interface do *switch* o projeto foi separado em três partes: *front-end*, *back-end* e banco de dados. O *front-end*, ou interface *web*, foi desenvolvido usando o *framework* para JavaScript VueJs e o *framework* Nuxt. O *back-end*, que funciona como mediador entre banco de dados e interface web, além de ser responsável por acessar e configurar os *switches* na rede local, foi desenvolvido utilizando o ambiente de execução JavaScript NodeJs, *a Application Programming Interface* (API) Puppeter e o *framework* NestJs. O banco de dados utilizado no desenvolvimento do projeto foi o MYSQL.

Palavras-chave: raspberry pi; comutador gerenciável; interface web; automação; redes de computadores.

ABSTRACT

The present work aims to present the development of a group of embedded systems software for the remote management of switches. The system was developed aiming at low cost when using a Raspberry Pi Model B +. For the development of the system, the web interface of a TP-Link TL-SG108E switch was studied. After studying the switch interface, the project was divided into three parts: front-end, back-end and database. The front-end, or web interface, was developed using the VueJs JavaScript framework and the Nuxt framework. The back-end, which works as a mediator between the database and the web interface, in addition to being responsible for accessing and configuring switches on the local network, was developed using the NodeJs JavaScript execution environment, the Application Programming Interface (API) Puppeter and the NestJs framework. The database used in the development of the project was MYSQL.

Keywords: raspberry pi; managed switch; web interface; automation; computer network.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
MAC	*Midia Access Control*
OSI	*Open Systems Interconnection*
OUI	*Organizationally Unique Identifier*
SBC	*Single Board Computer*
SMTP	*Simple Mail Transfer Protocol*
TCP	*Transfer Control Procotol*
UDP	*User Datagram Protocol*
WWW	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações iniciais

Um dos principais agentes da Terceira Revolução Industrial ou Revolução Tecnocientífica é a internet. A internet é uma rede de redes a nível global que possibilitou a aquisição e distribuição de grandes volumes de informação (TANENBAUM; WETHERALL, 2011).

A internet começou durante a Guerra Fria (1947-1991) quando o exército dos Estados Unidos viu a necessidade de enviar e receber informações entre suas bases espalhadas pelo país capaz de aguentar uma guerra nuclear. Nessa época, o Pentágono criou a *Advanced Research Projects Agency* (ARPA) para centralizar o orçamento de pesquisas de defesa. Na ARPA, Lerry Roberts, começou o desenvolvimento da ARPANET (HAFNER; LYON, 1998).

Para o desenvolvimento da ARPANET, os pesquisadores chamaram diversas universidades para se integrarem no sistema e fizeram contratos com empresas para a criação de dispositivos com os protocolos necessários para a rede (TANENBAUM; WETHERALL, 2011).

Após o fim da Guerra Fria, os Estados Unidos percebia que não poderia gerir sozinha a ARPANET e cedeu seu domínio. Com isso, a ARPANET foi se tornando a Internet (TANENBAUM; WETHERALL, 2011).

No dia de 22 de maio de 1973, um empregado da Xerox, Bob Metcalfe, descreve um sistema de rede capaz de conectar os computadores Xerox Altos, permitindo à esses aparelhos a comunicação entre si e também para impressoras a laser Este novo padrão usava cabo coaxial e, em sua primeira iteração, era chamada de "*thick* Ethernet", por causa de seu cabo que parecia uma mangueira de jardim, capaz de funcionar sem perdas a 500 metros e conectar até 100 aparelhos. Isso permitia uma maior distância de conexão, porém por ser rígido era difícil de implementar em edifícios que não eram adaptados a essa tecnologia (SPURGEON; ZIMMERMAN, 2013).

A *thin* Ethernet veio logo após a *thick* Ethernet, com um cabo mais fino e flexível, mas possuía menos capacidade (apenas trinta) de conexões e distância (185 metros) (TANENBAUM; WETHERALL, 2011).

A internet começou com linhas telefônicas dedicadas, por já existirem em diversas localidades. Porém com seu crescimento, outras maneiras de se conectar foram geradas, causando conflitos entre os protocolos existentes, o que forçou a criação de uma nova arquitetura de referência chamada de modelo de referência *Transfer Control Procotol* (TCP) / *Internet Protocol* (IP)(TANENBAUM; WETHERALL, 2011).

Com a popularização da internet, a *International Organization for Standardization* (ISO), ou Organização Internacional de Normalização criou o modelo de referência *Open Systems Interconnection* (OSI), um padrão criado com o intuito de padronizar a comunicação entre origem e destino, independente da tecnologia empregada na rede de computadores. Esse modelo se tornou importante para a visualização das várias camadas do modelo TCP/IP que compõem a

internet atual (TANENBAUM; WETHERALL, 2011).

Um dos primeiros dispositivos a serem implementados foram os *hub's*, que permitiam a comunicação entre os dispositivos da mesma rede, porém essa comunicação era feita de modo que um pacote de informação fosse enviado a todos os aparelhos na rede até encontrar seu destinatário, assim durante essa tarefa nenhum outro pacote poderia ser enviado. Esse método de comunicação tinha sérios problemas de segurança e também de congestionamento da rede, pois não permitia paralelismo de informação. Conforme aumentava a escala das empresas, os problemas dos *hub's* foram se agravando (SPURGEON; ZIMMERMAN, 2013).

O *switch* é um aparelho capaz de conectar diversos dispositivos utilizando a interface de rede Ethernet em uma rede física. O equipamento é capaz de paralelizar o envio de informação, pois ela trafega apenas entre os dispositivos de remetente e destino. Por conta disso, ele torna o envio de informação mais seguro dentro da rede. Esse dispositivo também apresenta outras funcionalidades como a qualidade de serviço que permite o controle de banda para níveis de importância de pacotes de informação (SPURGEON, 2014).

Um aparelho *switch* pode ser gerenciável (*Managed Switch*) ou não gerenciado (*Unmanaged Switch*). Um *switch* não gerenciado é mais simples e possui uma programação fixa, enquanto que um *switch* gerenciado possui mais funções e pode ser customizado (SPURGEON; ZIMMERMAN, 2013). O *switch* gerenciável pode ser programado através da conexão física na porta console ou, em alguns casos, uma interface *web* que pode ser acessada apenas pela rede local do aparelho.

O presente trabalho tem por objetivo desenvolver um sistema para a configuração de um ou mais *switches* gerenciáveis TL-SG108E fabricado pela TP-LINK. Esse objetivo é alcançado neste projeto ao mover a configuração dos dispositivos para um banco de dados remoto onde o sistema pode manter a configuração dos *switches* centralizada. Essa informação pode então ser consumida pelo *back-end* do sistema que está embarcado em uma *Single Board Computer* (SBC). As informações também podem ser visualizadas através da interface *web* para serem modificadas.

1.2 Objetivos

A seguinte Seção discorre sobre os objetivos gerais e específicos deste trabalho.

1.2.1 Objetivo geral

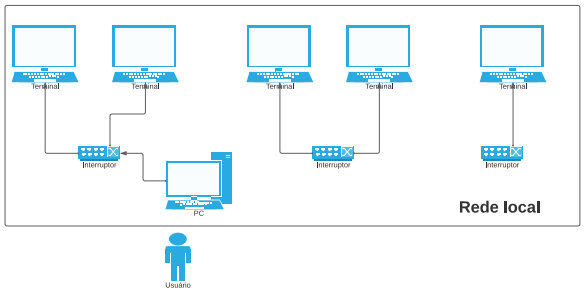
Projetar e desenvolver um sistema capaz de ser executado em em um sistema embarcado em uma SBC (Raspberry Pi, Orange Pi, Nano Pi, Beaglebone, etc) capaz de se comunicar com uma interface *web* remota onde será possível a gerência dos dispositivos cadastrados no sistema.

1.2.2 Objetivos específicos

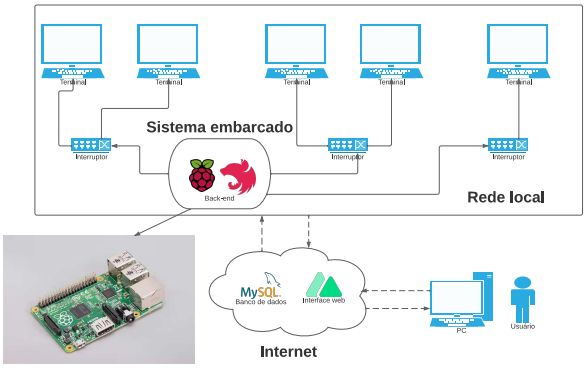
- Desenvolver um sistema *back-end* para gerência dos *switches*, que seja capaz de manipular as configurações de um *switch* gerenciável;
- Desenvolver uma interface *web* para controle das configurações de múltiplas unidades;
- Embarcar a solução desenvolvida em uma SBC.

1.3 Justificativa

Figura 1 – Configuração manual x configuração com o sistema
(a) Configuração manual



(b) Configuração com o sistema



Fonte: Autoria própria (2022)

Os *switches* gerenciáveis possuem uma variedade de preços decorrentes de suas funcionalidades. O TL-SG108E é considerado de baixo custo para pequenas empresas, custando em torno de U$29,99 e dispõe de 8 portas para conexão Ethernet, cotação feita em 12 de novembro de 2022 no site da Kabum (2022).

Com o intuito de se diminuir custos para empresas e instituições na infraestrutura de rede, é interessante a utilização de dispositivos com menos funcionalidades e menos automatização. Porém, ao passo em que a infraestrutura cresce, a manutenção da rede se torna custosa devido a difusão das configurações como um todo.

A difusão das configurações dos *switches* pela rede causa erros inesperados de difícil localização, como problemas de descoberta de rede ou problemas de velocidade de navegação entre pontos.

A Figura 1.a mostra uma rede com três *switches*, cada um desses *switches* deve ser acessado individualmente por um operador para saber qual a configuração atual do dispositivo e para mudar essa configuração.

Na Figura 1.b é mostrado o diagrama de funcionamento do sistema desenvolvido neste trabalho. Nessa dinâmica o *back-end* fica embarcado em uma SBC na rede local que se quer gerenciar. Enquanto que o banco de dados e a interface *web* ficam hospedados na Internet. O *back-end* serve tanto como um mediador entre o banco de dados e a interface *web*, quanto como um controlador dos *switches*. Nesse sistema, o usuário pode só acessar o sistema através da interface *web*. Assim, o sistema se mantém mais seguro, pois todas as requisições são verificadas pelo *back-end*.

A premissa desse trabalho se baseia na necessidade de conexão direta para configuração de um *switch* TL-SG108E através da interface *web* própria do aparelho e a escassa quantidade de solução para automatização desses dispositivos. Durante a confecção deste trabalho, não foram encontrados projetos que tivessem a capacidade de realizar a configuração automática de um *switch* TL-SG108E, tampouco de forma remota e centralizada. O projeto mais próximo encontrado foi o de Smode (2020) chamado de ”essstat”, um script em python criado para puxar as estatísticas das portas de um *switch* na rede local. Em busca de uma forma de se automatizar o processo, esse trabalho propõe o desenvolvimento de uma aplicação completa para controle remoto das configurações de *switches* em alta escala, permitindo a implantação de dezenas desses aparelhos e centralizando seu controle em uma SBC de baixo custo, com modelos como o Raspberry Pi 1 Model B que chega a custar U$25,00, cotação feita em 12 de novembro de 2022 nos sites da Digikey (2022) e Element 14 (2022).

2 REFERENCIAL TEÓRICO

A seguinte Seção apresenta uma introdução a, o que é um software de rede, o que é um modelo de camadas, aos modelos de camada OSI e TCP/IP e UDP e ao padrão Ethernet.

2.1 Software de rede e modelo de camadas

O *software* de rede é uma interface aplicada em uma rede de computadores, ele serve como língua franca em uma rede para comunicação entre os dispositivos. Já um modelo de camadas serve para separar um *software* de rede em pequenas partes, ao fazer essa divisão de responsabilidades é possível diminuir a complexidade da criação do *software* de rede (TANENBAUM; WETHERALL, 2011).

2.2 O modelo OSI

O modelo OSI veio como uma tentativa de se padronizar a Internet. O modelo OSI agrupa diversos protocolos em sete níveis abstratos, com funções distintas entre si, chamados de camada e elas são: física, enlace de dados, rede, transporte, sessão, apresentação e aplicação (TANENBAUM; WETHERALL, 2011).

A camada física define como os dados transitam pelo condutor, quantos pinos e suas finalidades, como é o sinal enviado, como se inicia e termina uma comunicação, velocidade de comunicação, se a comunicação é simultânea. A camada de enlace de dados recebe o sinal analógico, corrige se necessário, e envia para a camada de rede. Ela também é responsável pelo controle do fluxo de dados. A camada de rede conecta os protocolos de destino com a origem do sinal, agindo como mediador entre os protocolos de diferentes dispositivos. Também é dever da camada de rede o controle de tráfego dos pacotes existentes na rede. A camada de transporte garante a conexão entre camadas de rede distintas. Os protocolos da camada de sessão estabelecem uma comunicação entre dois dispositivos na rede. Essa camada oferece serviços de autenticação e sincronização entre os aparelhos de comunicação. A camada de apresentação se preocupa com a semântica e a sintaxe dos dados. A camada de aplicação contém os protocolos para os usuários finais, como o protocolo *HyperText Transfer Protocol* (HTTP) (TANENBAUM; WETHERALL, 2011).

2.3 O modelo TCP/IP e UDP

Segundo Tanenbaum e Wetherall (2011), o modelo TCP/IP e *User Datagram Protocol* (UDP) é o padrão mais utilizado no mundo, sendo o primeiro modelo a ser criado para a *World Wide Web* (WWW). Esse modelo tem as seguintes camadas: enlace de dados, rede, transporte

e aplicação.

A camada de enlace é uma série de protocolos que agem como uma interface para conectar os *hosts* na rede. A camada de rede ou camada de Internet contém a arquitetura de toda a rede de Internet. A camada de transporte é a responsável por conectar origem e destino, separando a informação em blocos e remontando no destino. Na camada de transporte ficam os protocolos TCP e UDP. O TCP é mais seguro, feito para conexões confiáveis. Já o UDP não é confiável, mas permite maior flexibilidade para controle de fluxo. Na camada de aplicação está o controle de protocolos de alto nível voltado para o usuário final como os protocolos de transferência de arquivos *File Transfer Protocol* (FTP) e de correio eletrônico *Simple Mail Transfer Protocol* (SMTP) (TANENBAUM; WETHERALL, 2011).

2.4 Padrão Ethernet

O padrão Ethernet é um padrão de comunicação, patenteado em 1976 como um "Sistema de Comunicação de Dados Multiponto com Detecção de Colisão" Spurgeon e Zimmerman (2013). Ele é divido em três partes: o quadro Ethernet, o *Midia Access Control* (MAC) e o meio físico.

2.4.1 Quadro Ethernet

No quadro ou *frame* de Ethernet é onde estão os dados a serem movidos de um dispositivo ao outro. Considerando uma conexão Ethernet de 10MB/s *half-duplex*[1] compartilhada, o quadro é formado por no minimo 66 *bytes* e no máximo 1520 *bytes*(SPURGEON; ZIMMERMAN, 2013).

Como mostrado no quadro 2, o quadro começa com 8 *bytes* de preâmbulo indicando que começou o envio dos dados, o padrão de *bits* é a seguinte: os seis primeiros *bits* são 101010 e os dois últimos são 11. (TANENBAUM; WETHERALL, 2011, p. 177). Em seguida, existem 6 bytes contendo o endereço de destino e endereço da fonte do *frame*. A seguir existem 2 *bytes* para identificar o tipo de protocolo de alto nível em que a informação está codificada. Depois dos *bytes* de tipo, vem os *bytes* de informação, podendo variar de 46 até 1500 *bytes*. Por último, existem 4 *bytes* que sinalizam o fim da informação e checagem da integridade dos dados enviados.

Figura 2 – Quadro Ethernet

Preambulo	Endereços	Tipo	Informação	Integridade
8 *bytes*	6 *bytes*	2 *bytes*	46 - 1500 *bytes*	4 *bytes*

Fonte: Autoria própria (2022)

[1] *Half-duplex* significa que apenas um computador pode enviar dados pela conexão Ethernet por vez.

2.4.2 Tamanho mínimo do quadro

Segundo Tanenbaum e Wetherall (2011) a Ethernet clássica de 10MB/s, em seu comprimento máximo de 2500 m e com quatro repetidores – limite máximo para se recuperar o sinal – possui um tempo de propagação de 25 ms, sendo necessário 50 ms para que o sinal chegue ao destinatário e volte ao transmissor. Por conta desse atraso é necessário que exista um tamanho mínimo dos *bytes* de informação do quadro Ethernet para que um dispositivo não termine de transmitir a mensagem antes que o primeiro *bit* de informação alcance o destinatário, caso contrário a mensagem seria perdida e o remetente acreditaria que chegou ao destino.

2.4.3 Protocolo CSMA/CD

Na Ethernet clássica todos os dispositivos se conectam ao mesmo barramento de comunicação. Quando duas máquinas enviam simultaneamente uma mensagem no barramento, essas informações se embaralham e são corrompidas, causando a chamada colisão (TANENBAUM; WETHERALL, 2011).

O protocolo *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD) foi desenvolvido para impedir que colisões aconteçam. Nesse protocolo todas as estações permanecem escutando o barramento, quando uma máquina decide enviar uma mensagem ela verifica se tem algum sinal no barramento, caso não exista nenhum sinal ela começa a enviar seus dados, caso ela não receba uma confirmação de integralidade de seus dados, ela assume que ocorreu uma colisão e tenta novamente em outro momento (TANENBAUM; WETHERALL, 2011).

2.4.4 Protocolo MAC

O endereço MAC é obtido ao se juntar 24 *bits* da identificação única do fabricante o *Organizationally Unique Identifier* (OUI) e 24 *bits* únicos para o dispositivo Ethernet (SPURGEON; ZIMMERMAN, 2013).

Na Ethernet *half-duplex* clássica todas as máquinas de uma rede são independentes umas das outras, porém todas elas estão conectadas á mesma rede de dados compartilhada. Ao se enviar um sinal na rede, todos os dispositivos recebem o sinal, ao receber um *frame*, a estação verifica o campo de destino e o compara ao próprio endereço MAC, se não for igual a informação é descartada (SPURGEON, 2014).

2.4.5 Camada física

Como visto na seção 2.3, a camada física é a responsável por padronizar o tipo de sinal e o meio onde esse sinal ira trafegar, trabalhando em conjunto com o protocolo MAC, essa camada é sensível as mudanças desse protocolo (SPURGEON; ZIMMERMAN, 2013).

2.5 Ethernet Comutada

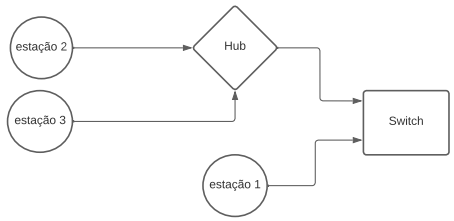
Na Ethernet comutada cada estação tem seu próprio cabo que se conecta ao *hub* que faz o papel do canal de dados compartilhado da Ethernet clássica (TANENBAUM; WETHERALL, 2011).

Esse novo tipo de Ethernet recebe esse nome por causa dos aparelhos *switches* (comutadores), externamente os *switches* são equivalentes aos *hubs*. Os dois interligam estações de trabalho a uma única rede, porém o diferencial do *switch* é conter internamente uma placa que redireciona as chamadas dos quadros para apenas o destino da mensagem, ao invés de enviar para toda a rede, utilizando sua tabela de endereçamento. Esse processo é totalmente transparente para a rede, todas as outras estações não sabem que o quadro foi enviado (SPURGEON; ZIMMERMAN, 2013).

2.5.1 Colisões

Diferente dos *hubs* ou da Ethernet clássica, onde todas as estações estão em uma mesma rede compartilhada e por isso podem sofrer colisões. A Ethernet comutada funciona em *full-duplex*, ou seja, tanto o *switch*, quanto a estação podem enviar simultaneamente dados as outras estações, portanto, não sofre com colisões, porém caso dois dispositivos tentem enviar uma mensagem à mesma porta, o comutador deve enfileirar os quadros no *buffer* interno (SPURGEON; ZIMMERMAN, 2013).

Figura 3 – Esquema de rede com *switch* e *hub*



Fonte: Autoria própria (2022)

Switches e *hubs* podem ser cascateados como na figura 3 (SPURGEON, 2014), enquanto as estações 2 e 3 disputam a rede do *hub* como na internet clássica com colisões e espera, ao chegar no *switch* ela será tratada como quadro normal e redirecionado a porta soli-

citada.

2.6 Sistema embarcados

Por sua vasta gama de dispositivos disponíveis no mercado, os sistemas embarcados estão presentes na maioria das coisas do dia a dia. Com a popularização da internet das coisas, aparelhos como geladeiras, micro-ondas e lâmpadas estão se "tornando inteligentes", com a capacidade de se comunicar com outros dispositivos, podendo ser automatizados ou possibilitando uma configuração personalizada e não mais engessada pelo fabricante (ALMEIDA; MORAES; SERAPHIM, 2017).

A definição do que é um sistema embarcado é tema de debates na área, mas para Almeida, Moraes e Seraphim (2017) os sistemas embarcados são dispositivos eletrônicos com *hardware* e *software* montados para uma especificação particular.

Segundo Almeida, Moraes e Seraphim (2017), um sistema embarcado não pode ser considerado um computador, pois ele é feito sob medida para a tarefa que deve executar. Um sistema embarcado é incapaz de executar qualquer outra função além da qual foi projetado para realizar.

2.6.1 Formas de funcionamento de um sistema embarcado

Um sistema embarcado pode ser reativo ou controlado por tempo. Quando reativo, o sistema espera que uma entrada ou mudança ocorra para executar uma tarefa. Ele ainda pode ser controlado por tempo, onde a tarefa é executada em intervalos definidos de tempo (ALMEIDA; MORAES; SERAPHIM, 2017).

Para o funcionamento por controle de tempo ela pode ser crítica ou não. Em sistema de controle de tempo crítico, a tarefa deve ser executada exatamente no tempo pré estabelecido (ALMEIDA; MORAES; SERAPHIM, 2017).

2.6.2 Raspberry Pi

O Raspberry Pi é uma SBC por conter em uma placa de 8,5 cm por 5,6 cm. Esse dispositivo é empregado para projetos de sistemas embarcados, por ser capaz de executar tarefas como um computador, ao passo em que é mais barato, pequeno e com um consumo menor de energia quando comparado a um computador padrão. Esse mini computador possui pinos de conexão capazes de acionar atuadores ou ler sensores em campo. Além de conectores Ethernet, USB e HDMI que permitem construir um ambiente similar ao computador de mesa tradicional (Raspberry Pi, 2022). A Raspberry Pi 1 Model B + foi escolhida para esse trabalho, por sua disponibilidade, mas ela também possui outras características favoráveis, como seu

tamanho e seu preço que varia de U$25,00 e U$45,00, cotação feita em 12 de novembro de 2022 em (DIGIKEY, 2022) e (Element 14, 2022).

3 MATERIAIS E MÉTODO

O presente capítulo discorre sobre os materiais e métodos utilizados na execução do projeto.

3.1 Materiais

Para o desenvolvimento do projeto, foi utilizado o banco de dados MYSQL, o ambiente de execução NodeJS utilizando o *framework* NestJS e a biblioteca VueJS com o framework Nuxt para interface de usuário *web*. O dispositivo escolhido para embarcar essa solução foi um Raspberry Pi Model B+ que pode ser vista na Figura 4.

Figura 4 – Raspberry Pi Model B+

Fonte: Raspberry Pi (2022, p.1)

Para abranger todas as funcionalidades necessárias para o projeto, a interface *web* de um *switch* TL-SG108E foi estudada para mapear as suas funções para o desenvolvimento das tabelas e funcionalidades do programa.

3.2 Método

O desenvolvimento desse projeto procura implementar um sistema remoto para o gerenciamento de diversos *switches* em uma rede e embarcar esse sistema em uma SBC seguindo o diagrama mostrado na figura 1.b.

Para o funcionamento do sistema é necessário a instalação do ambiente de execução de NodeJs na versão 14.17.1. Enquanto que para a interface *web* o site é gerado estaticamente, contendo todos os arquivos *HyperText Markup Language* (HTML) necessários.

O desenvolvimento pode ser separado em três partes: *front-end*, *back-end* e banco de dados. O *front-end* é a interface *web* do sistema, nele consta as informações dos *switches*. O *back-end* é responsável por prover informações para o *front-end* através de requisições HTTP, atualizar os *switches* na rede e também serve como uma barreira de proteção contra ações mal intencionadas ao banco de dados. O banco de dados guarda toda a informação dos *switches*.

O *front-end* foi desenvolvido de forma que possua todas as informações pertinentes mostradas ao usuário, buscando a menor quantidade de cliques para chegar a informação de-

sejada. O *back-end* abstrai toda a regra de negócio do projeto para se ter requisições HTTP diretas e simples, minimizando a lógica no *front-end*. O banco de dados foi moldado de acordo com a leitura lógica da interface web do *switch*.

É possível visualizar na Figura 5, a tela de login da interface web do dispositivo. Outras telas estão disponíveis no Apêndice A.

Figura 5 – Interface de login do *switch* TL-SG108E

Fonte: Autoria própria (2022)

4 RESULTADOS

Este capítulo informa sobre o sistema desenvolvido e os resultados de sua execução.

4.1 Escopo do sistema

O sistema desenvolvido consiste em duas partes, *front-end* (interface com o usuário) e *back-end* (API). No *front-end*, o usuário pode visualizar as informações dos *switches* cadastrados no sistema e mudar suas configurações, ao selecionar um *switch* o usuário é levado a uma tela onde pode mudar individualmente as configurações das portas daquele aparelho. O *front-end* não tem acesso ao banco de dados diretamente, tendo como intermediário o *back-end*, que responde suas requisições HTTP e trata os dados que vão e que saem do banco de dados.

Já no *back-end*, as requisições do usuário são recebidas através de uma requisição HTTP. Os *endpoints*, que são endereços em que a API está escutando, do servidor permitem recuperar informações como configurações gerais do *switch*, configurações atuais das portas, configurações possíveis para status, velocidade máxima, e controle de fluxo. Também é possível atualizar informações do *switch* e suas configurações de porta. Além disso, o servidor também é responsável por uma rotina contínua que é executada em um intervalo de tempo pré-definido em que varre o banco de dados e atualiza os *switches* da rede com base nas informações recuperadas do banco.

4.2 Modelagem do sistema

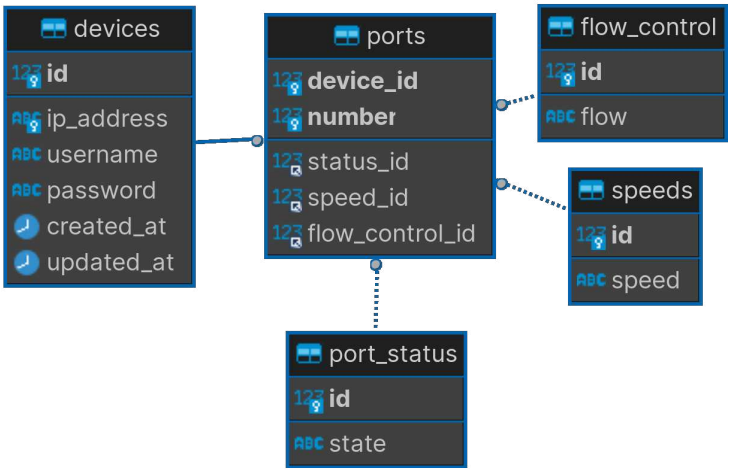
4.2.1 Banco de dados

O banco de dados é a memória do sistema, sua modelagem pode ser visto na Figura 6, o banco de dados possui cinco tabelas: *devices*, *ports*, *flow_control*, *speeds* e *port_status*.

A tabela *devices* contém a informação de criação e atualização, endereço IP da máquina e usuário e senha para acesso a interface *web* do dispositivo. Na tabela *ports* estão as informações de cada porta de um dado *switch*, tendo como chave primária o id do *switch* e o número da porta a que se refere, além das informações de controle de fluxo de dados, velocidade e status da porta. As tabelas *flow_control*, *speeds* e *port_status* se referem a uma tabela para enumeração dos possíveis estados, essas tabelas são necessárias para que os dados permaneçam homogêneos e consistentes dentro do banco de dados.

O modelo do banco de dados foi implementado de forma a facilitar a adição de novas funcionalidades ao sistema.

Figura 6 – Modelo do banco de dados



Fonte: Autoria própria (2022)

4.2.2 *Back-end*

No *back-end* estão as abstrações do sistema, o *back-end* é responsável por receber requisições HTTP para recuperar ou alterar informações do banco e devolver essas informações tratadas para o *front-end*, além de ser responsável por manter a consistência entre as configurações armazenadas no banco de dados e os *switches* da rede. Através de seus *endpoints* é possível adicionar *switches* e alterar e suas portas. As Figuras 7 e 8 mostram o fluxograma da lógica para algumas requisições do sistema.

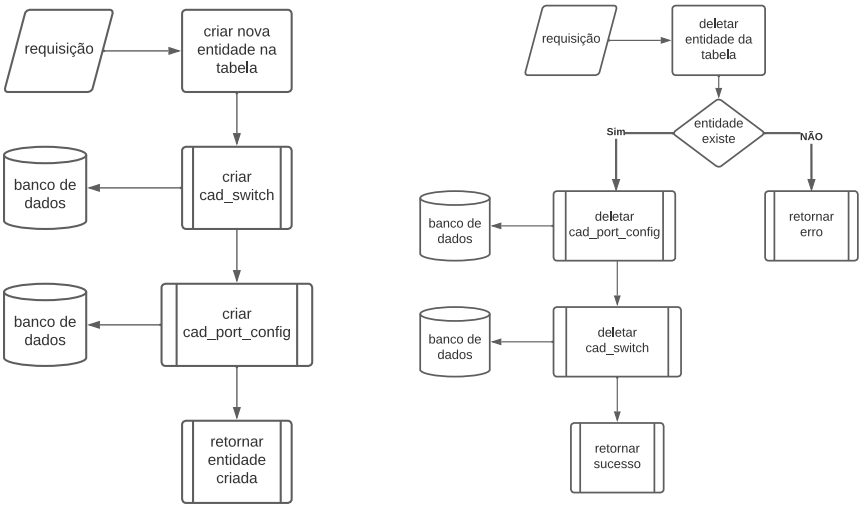
O fluxograma da Figura 7.a mostra como o servidor processa uma requisição de criação de um novo *switch*, nessa requisição o servidor gera uma transação onde tenta adicionar um novo dispositivo e portas no banco de dados, para posteriormente vincular-se ao dispositivo físico, caso ocorra um erro o servidor informa ao usuário. Já na Figura 7.b tem-se o fluxo para a remoção de um *switch*, onde se inicia uma transação para remover tanto as portas quanto o dispositivo, no caso de falha é mandado uma resposta de falha ao usuário. Essa transação é necessária para caso uma das consultas retornem um erro o banco de dados não sofre nenhuma alteração, e continua consistente. Esse trecho e outros podem ser consultados no Apêndice B.

No fluxograma presente na Figura 8.a é demonstrada a ação de atualizar um dispositivo no banco de dados. Já Figura 8.b mostra como é o fluxo para atualizar as configurações de um *switch* na rede, que recupera as informações contidas no banco de dados, tenta se conectar ao *switch* relacionado e então aplicar no dispositivo as novas configurações.

O código fonte do sistema está disponível através do link https://github.com/andrelmd.

Figura 7 – Fluxograma de criação e remoção

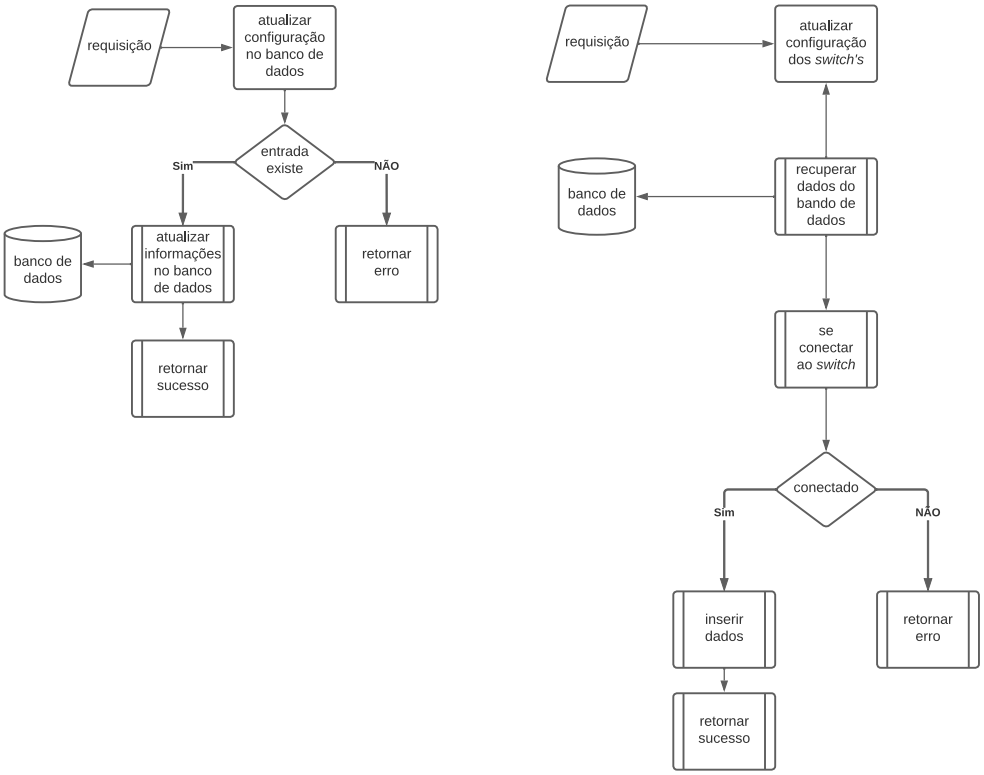
(a) Criação de entidade **(b) Remoção de entidade**



Fonte: Autoria própria (2022)

Figura 8 – Fluxogramas de atualização

(a) Atualização de banco de dados **(b) atualização de *switch***



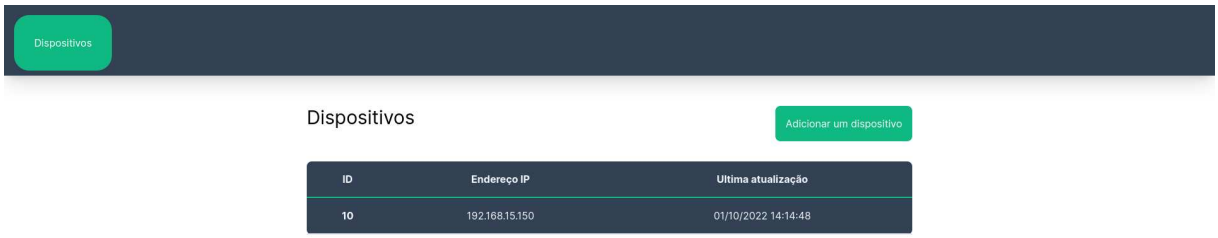
Fonte: Autoria própria (2022)

4.2.3 *Front-end*

O *front-end* é a parte mais importante do sistema ao que compete o usuário. É através da interface *web* que o usuário interage com o sistema. Toda a informação que não está sendo tratada por ele simplesmente não faz parte do sistema.

A Figura 9 mostra uma tela onde é possível ver todos os dispositivos cadastrados no sistema em uma lista. Ao clicar em um dispositivo cadastrado o usuário é levado para a tela na Figura 10 onde pode modificar as configurações do *switch*. Também é possível adicionar um novo dispositivo ao clicar no botão ”Adicionar um dispositivo” na tela da Figura 9 que leva à tela apresentada na Figura 11 onde o usuário pode adicionar um novo dispositivo.

Figura 9 – Tela de dispositivos



Fonte: Autoria própria (2022)

4.2.4 Testes realizados

O desenvolvimento do projeto começou com o *back-end*, para fazer os testes em cada *endpoint*, foi utilizado o Postman, um programa para facilitar requisições HTTP para testes em API, um exemplo de teste pode ser visto na Figura 12.

Os testes no *front-end* foram feitos manualmente, ou seja, a cada implementação de funcionalidade na interface web é feito um teste manual com cliques ou inserção de dados para verificar se a entrada de dados está correta.

Para os testes com a automatização da configuração dos *switches* utilizando a API Puppeteer foi necessário verificar os comportamentos da interface web ao navegar por ela, assim criando um *script* para realizar essas ações.

Durante os testes com a automatização foi identificado um problema de *timeout* (tempo

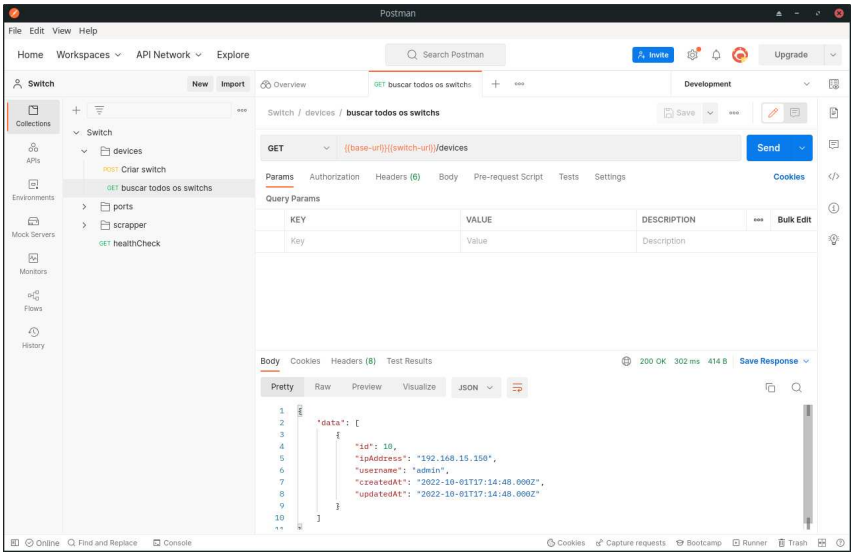
Figura 10 – Tela de portas

Fonte: Autoria própria (2022)

Figura 11 – Tela de adicionar novo dispositivo

Fonte: Autoria própria (2022)

Figura 12 – Exemplo de teste da API utilizando o Postman



Fonte: Autoria própria (2022)

esgotado), esse erro ocorre quando o serviço está tentando navegar de uma tela para outra, mas o navegador controlado não consegue renderizar a tela no tempo estipulado. Esse problema pode ser contornado ao se aumentar o *timeout* para navegação ou mudar para uma SBC com maior capacidade de processamento. Esse problema pode ocorrer, também, por uma conexão lenta entre SBC e *switch*.

5 CONCLUSÃO

Esse trabalho, tem como objetivo desenvolver um sistema embarcado de código aberto para gerência unificada e automatizada de *switches* TL-SG108E, sendo composto de servidor, banco de dados e interface *web*, como utilitário para o usuário final e, também, como fonte de consulta para sistemas que interagem com equipamentos via requisições HTML.

Este trabalho apresenta uma solução para múltiplas configurações remotas do *switch* TL-SG108E que se utiliza de uma SBC. Essa solução estende a gerência de switches para fora da rede local, permitindo uma centralização da configuração da infraestrutura de rede

O presente trabalho desenvolve um sistema embarcado capaz de centralizar a configuração de diversos switches em um banco de dados e possibilitando o acesso a essas informações de forma remota. Também foi desenvolvida uma interface *web* para a comunicação entre usuário e sistema. Por fim, essa solução foi embarcada em uma SBC para ser implementada em qualquer rede local com acesso ao *switch* do usuário.

Para o desenvolvimento do sistema, a interface web e as funcionalidades do *switch* foram mapeadas. Através disso, o sistema foi projetado para acomodar as funções do *switch*.

Para mudar as configurações dos *switches* na rede local, o servidor embarcado utiliza do Puppeteer para acessar o aparelho através de um navegador de internet, simulando um usuário. Essa ferramenta pode navegar através da interface utilizando as *tags* HTML, usadas para montar os sites, para interagir com o sistema ao simular cliques e entradas de dados no navegador.

O código fonte completo esta disponível através do link https://github.com/andrelmd.

REFERÊNCIAS

ALMEIDA, R. M. A. de; MORAES, C. H. V. de; SERAPHIM, T. d. F. P. **Programação de Sistemas Embarcados: Desenvolvendo Software para Microcontroladores em Linguagem C**. [*S.l.*]: Elsevier Brasil, 2017.

CHOPIN SEBASTIEN CHOPIN, P. P. A. **Nuxt.js**. 2013. https://nuxtjs.org/.

DEVELOPERS, L. F. N. **Node.js**. 2009. https://nodejs.org/.

DIGIKEY. **RASPBERRY PI B+**. 2022. Disponível em: https://www.digikey.com.br/en/products/detail/raspberry-pi/RASPBERRY-PI-B/6235381?src=raspberrypi. Acesso em: 12 de novembro de 2022.

Element 14. **Meet the Raspberry Pi 4 Model B!** 2022. Disponível em: https://community.element14.com/products/raspberry-pi/. Acesso em: 12 de novembro de 2022.

HAFNER, K.; LYON, M. **Where wizards stay up late: The origins of the Internet**. [*S.l.*]: Simon and Schuster, 1998.

KABUM. **Switch TP-Link 10/100/1000 Mbps, Easy Smart Gigabit, 8 Portas - TL-SG108E**. 2022. Disponível em: https://www.kabum.com.br/produto/129448/switch-tp-link-10-100-1000-mbps-easy-smart-gigabit-8-portas-tl-sg108e. Acesso em: 12 de novembro de 2022.

LLC, G. **Puppeteer: Headless Chrome Node.js API**. 2017. https://github.com/puppeteer/puppeteer.

MYSLIWIEC, K. **Nest.js**. 2017. https://nestjs.com/.

ONION, A.; SULLIVAN, M.; MULLEN, M. **The Invention of the Internet**. 2010. https://www.history.com/topics/inventions/invention-of-the-internet.

Raspberry Pi. **Buy a Raspberry Pi 1 Model B+**. 2022. Disponível em: https://www.raspberrypi.com/products/raspberry-pi-1-model-b-plus/. Acesso em: 24 de outubro de 2022.

SMODE, P. **PSMODE/ESSSTAT: TP-link easy smart switch port statistics**. 2020. Disponível em: https://github.com/psmode/essstat.

SPURGEON, C. **Ethernet: The Definitive Guide**. 2. ed. Sebastopol, CA: O'Reilly Media, 2014.

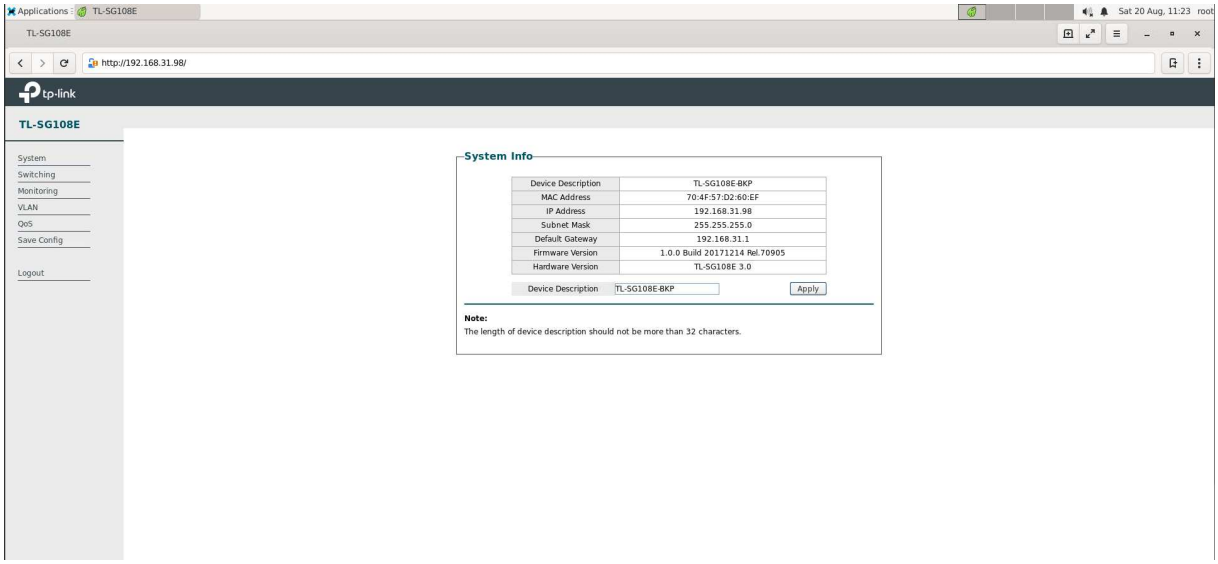
SPURGEON, C. E.; ZIMMERMAN, J. **Ethernet switches: An introduction to network design with switches**. [*S.l.*]: "O'Reilly Media, Inc.", 2013.

TANENBAUM, A.; WETHERALL, D. **Redes de computadores**. [*S.l.*]: Pearson Prentice Hall, 2011. ISBN 9788576059240.

YOU, E. **VueJs**. 2013. https://vuejs.org/.

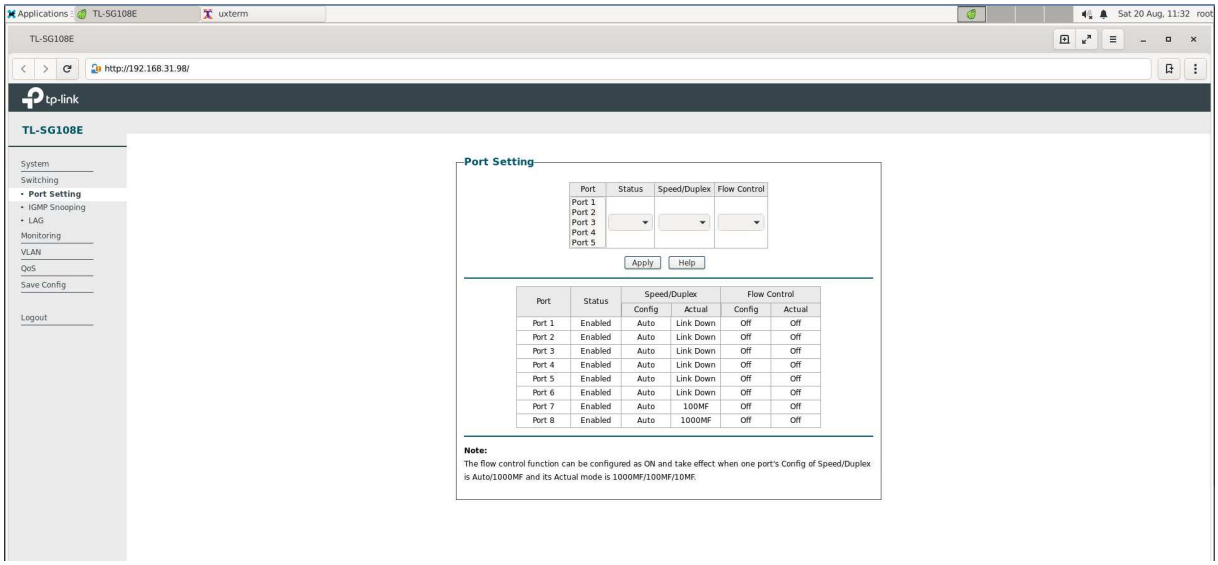
APÊNDICE A – Interface web do *switch*

Figura 13 – Informações do sistema do *switch*



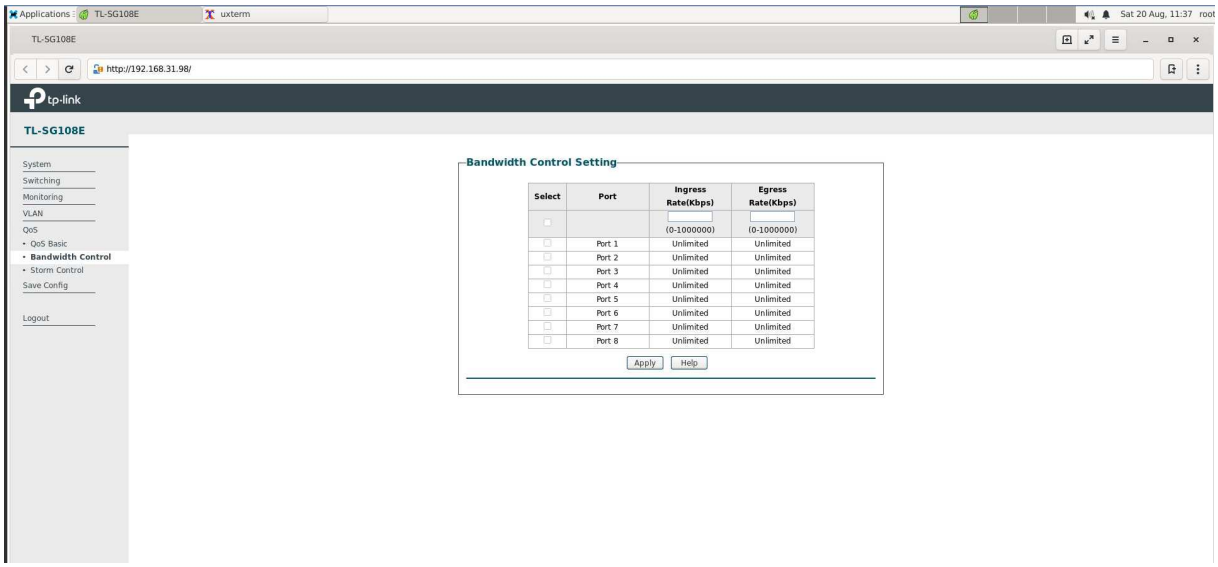
Fonte: Autoria própria (2022)

Figura 14 – Configuração e estado atual das portas



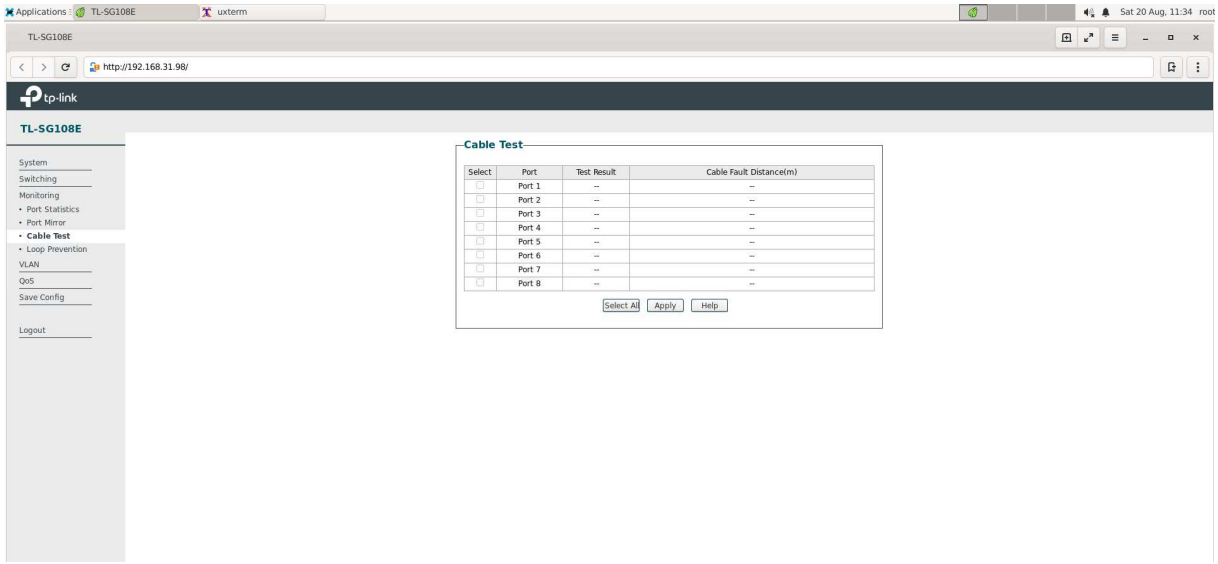
Fonte: Autoria própria (2022)

Figura 15 – Largura de banda



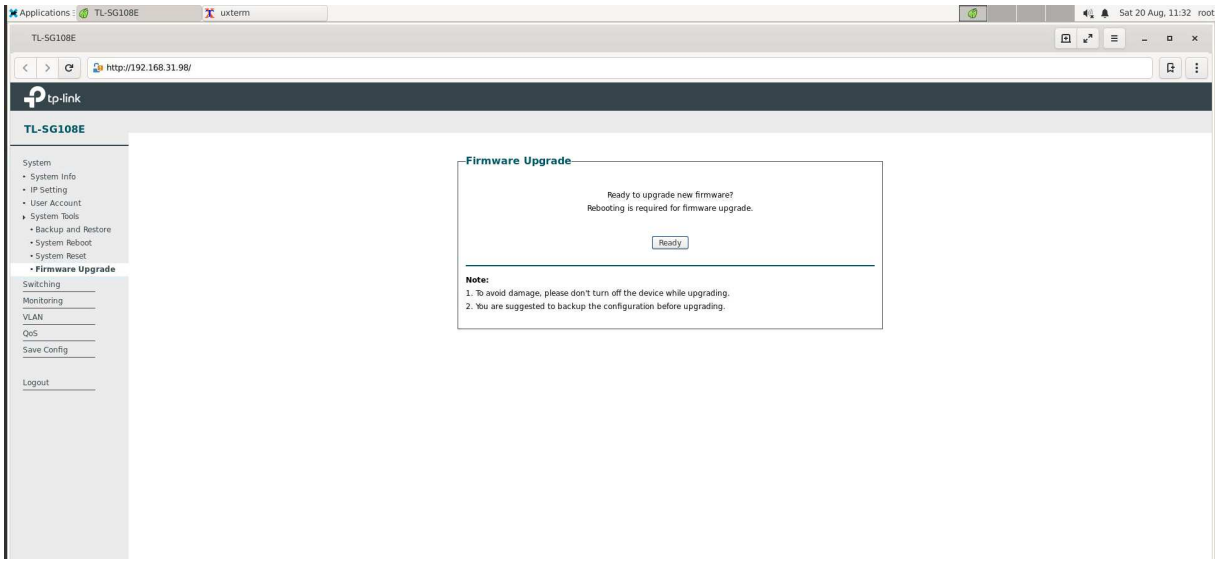
Fonte: Autoria própria (2022)

Figura 16 – Teste de cabos



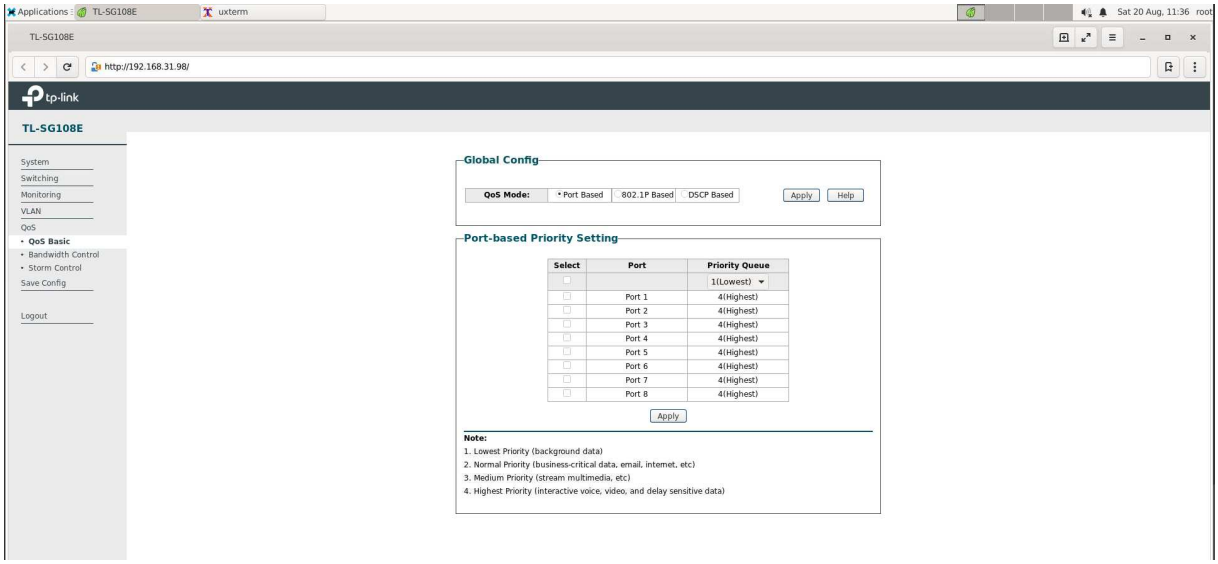
Fonte: Autoria própria (2022)

Figura 17 – *Upgrade* de *firmware*



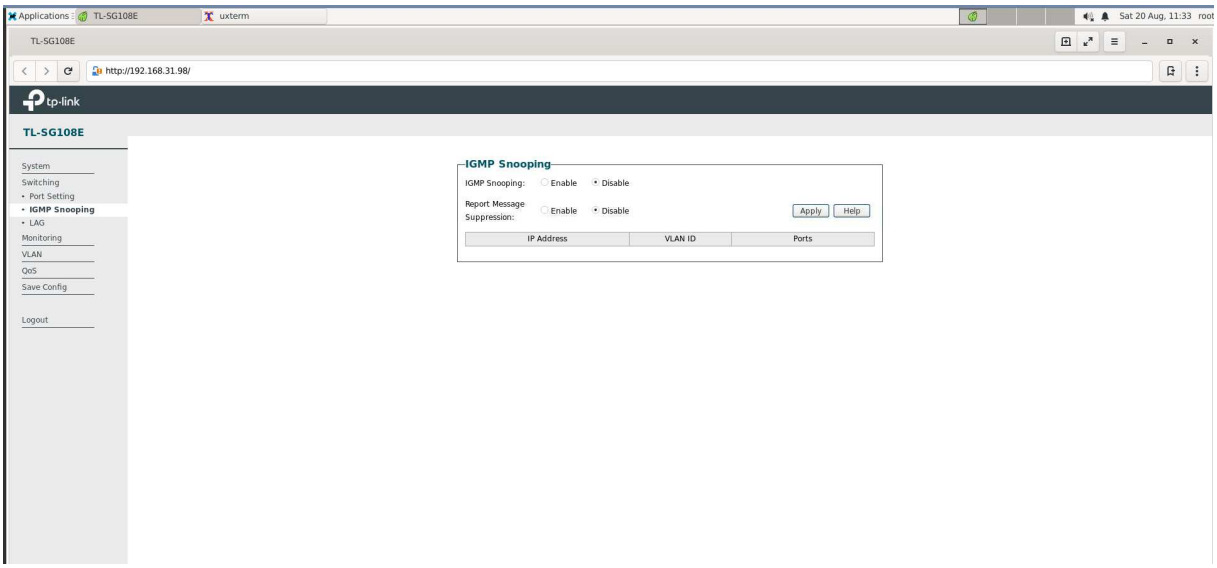
Fonte: Autoria própria (2022)

Figura 18 – Configuração global



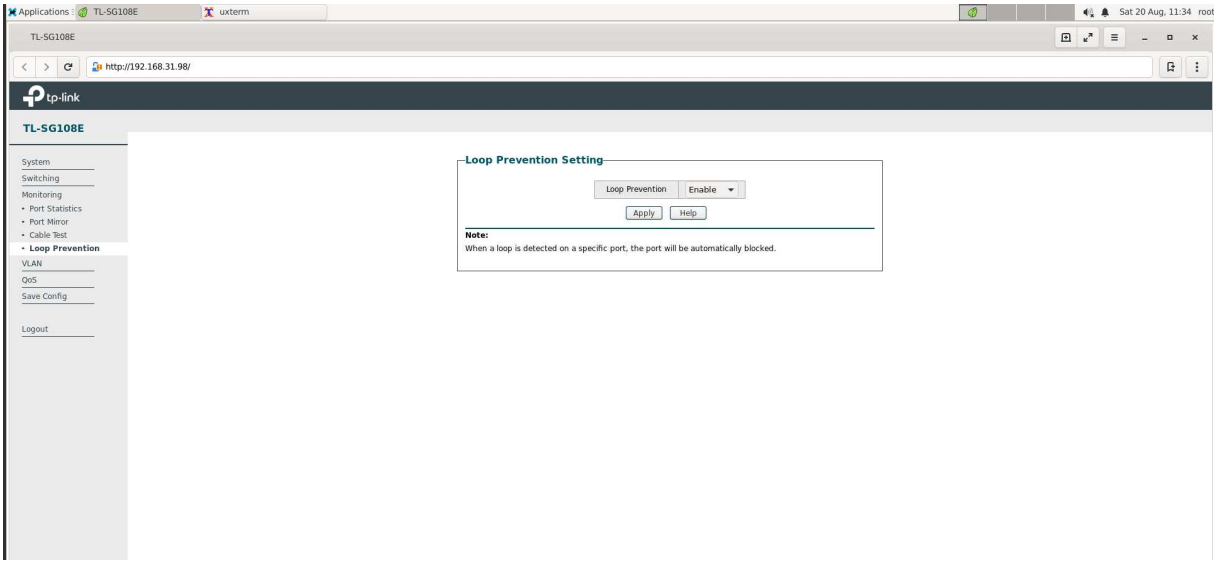
Fonte: Autoria própria (2022)

Figura 19 – Configuração IGMP



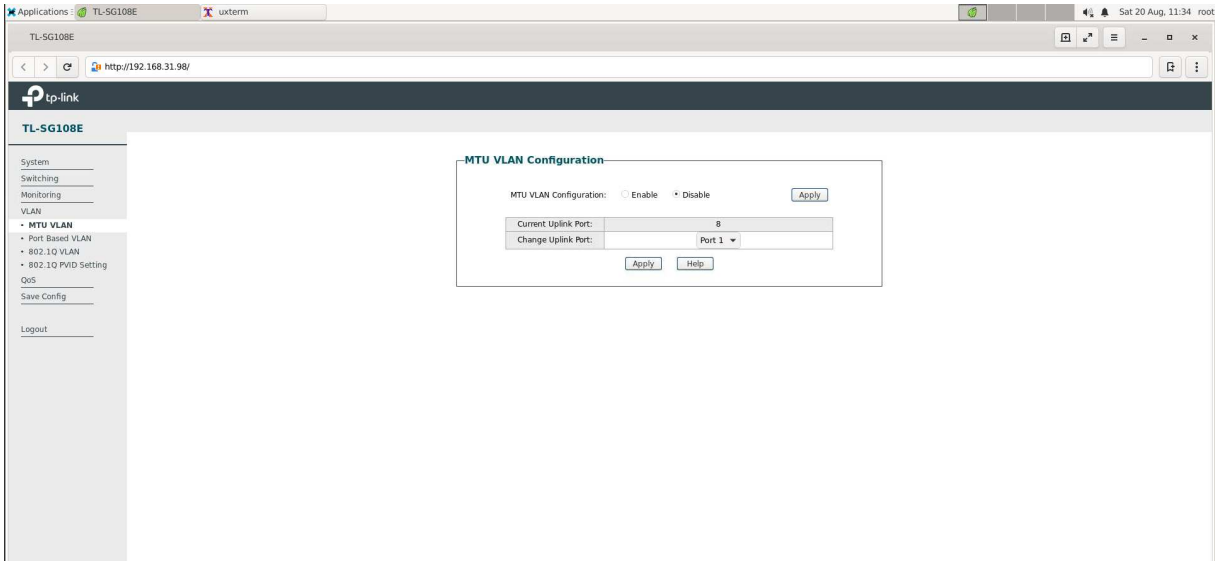
Fonte: Autoria própria (2022)

Figura 20 – Configuração prevenção de *loop* de sinal



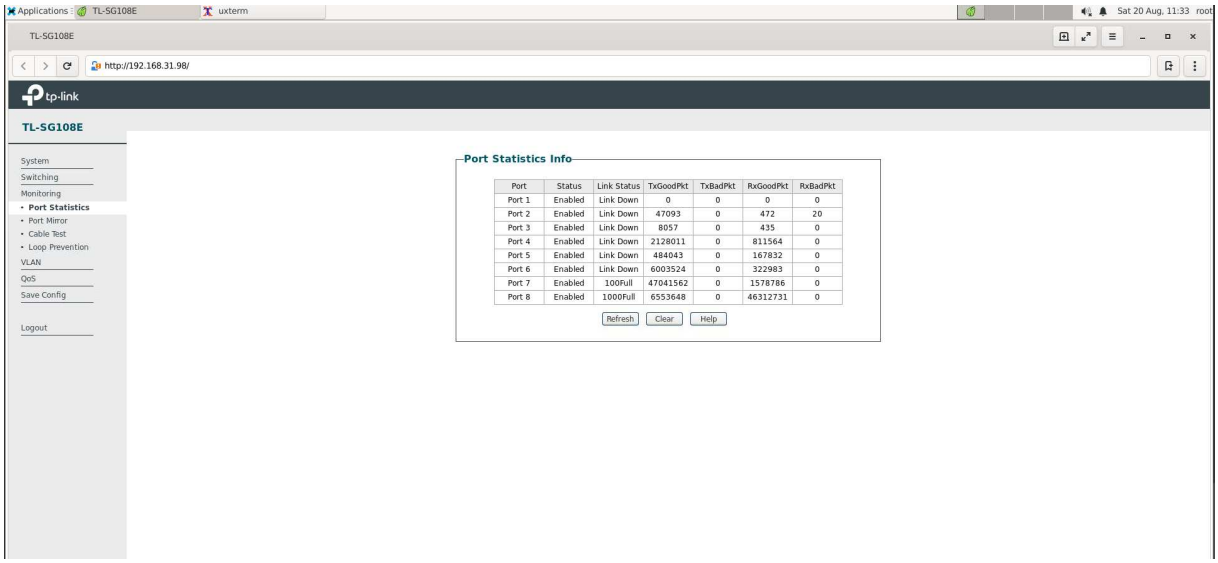
Fonte: Autoria própria (2022)

Figura 21 – Configuração de VLAN MTU



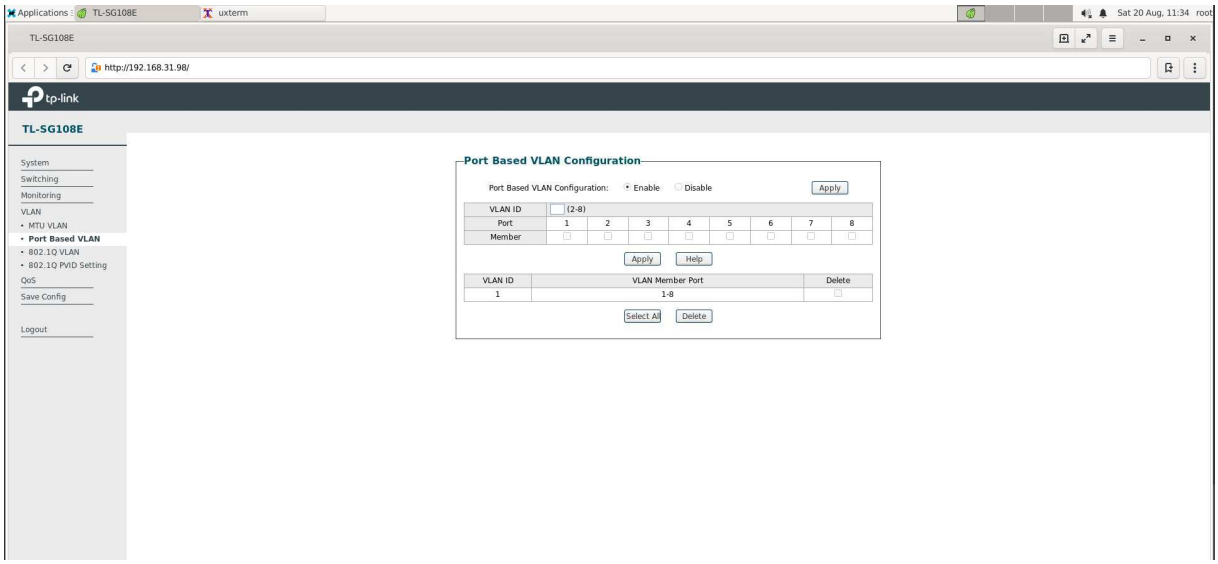
Fonte: Autoria própria (2022)

Figura 22 – Estatísticas das portas



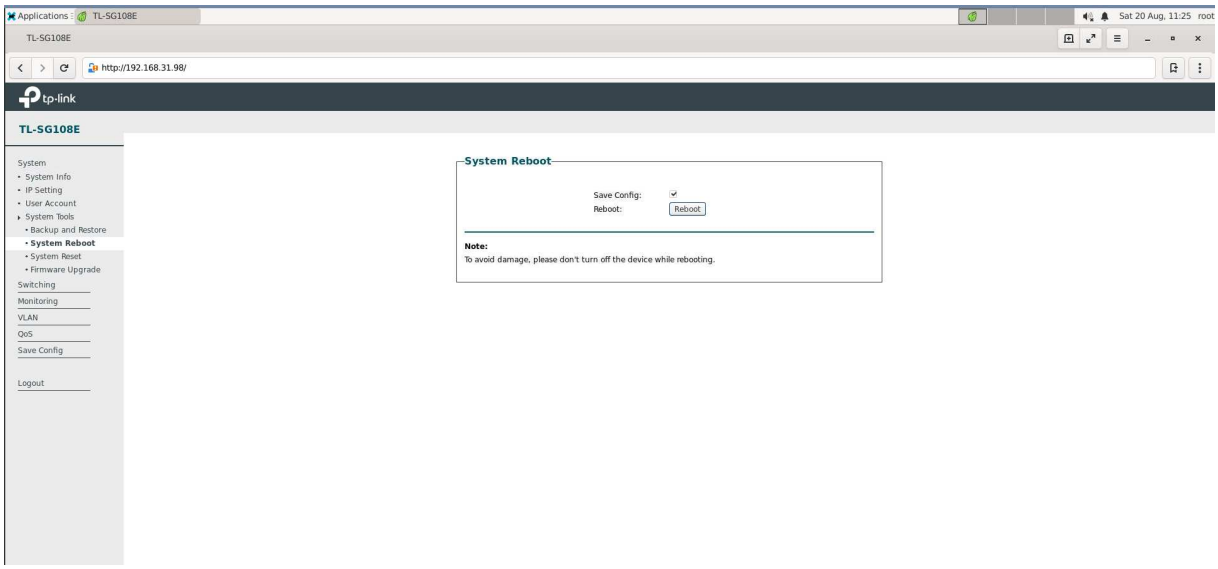
Fonte: Autoria própria (2022)

Figura 23 – Configuração de VLAN por entre as portas



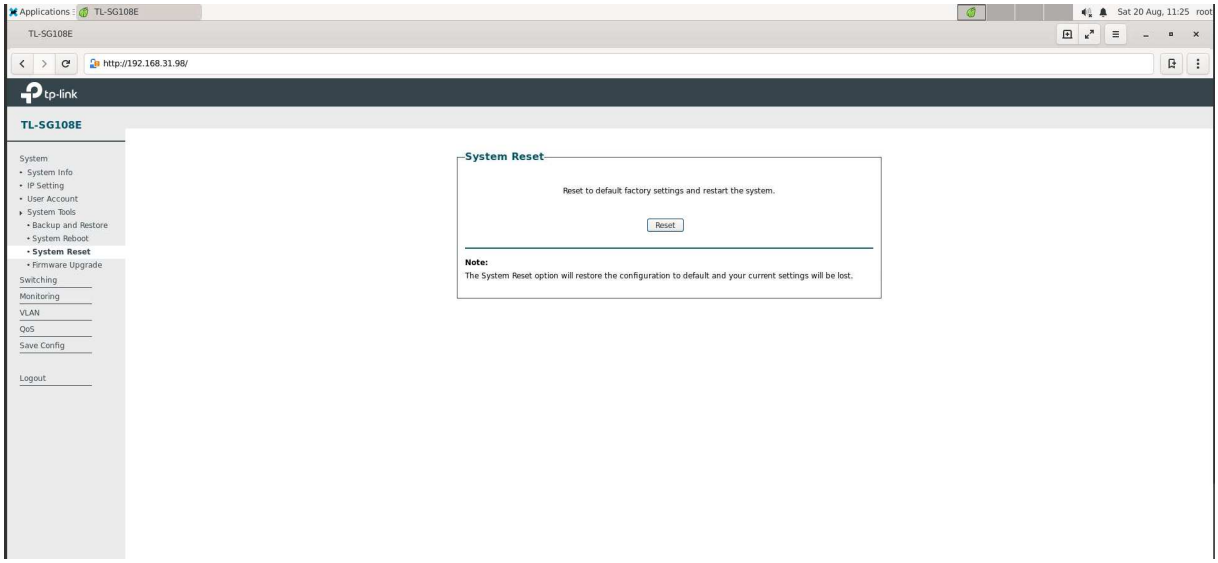
Fonte: Autoria própria (2022)

Figura 24 – Reiniciar *switch*



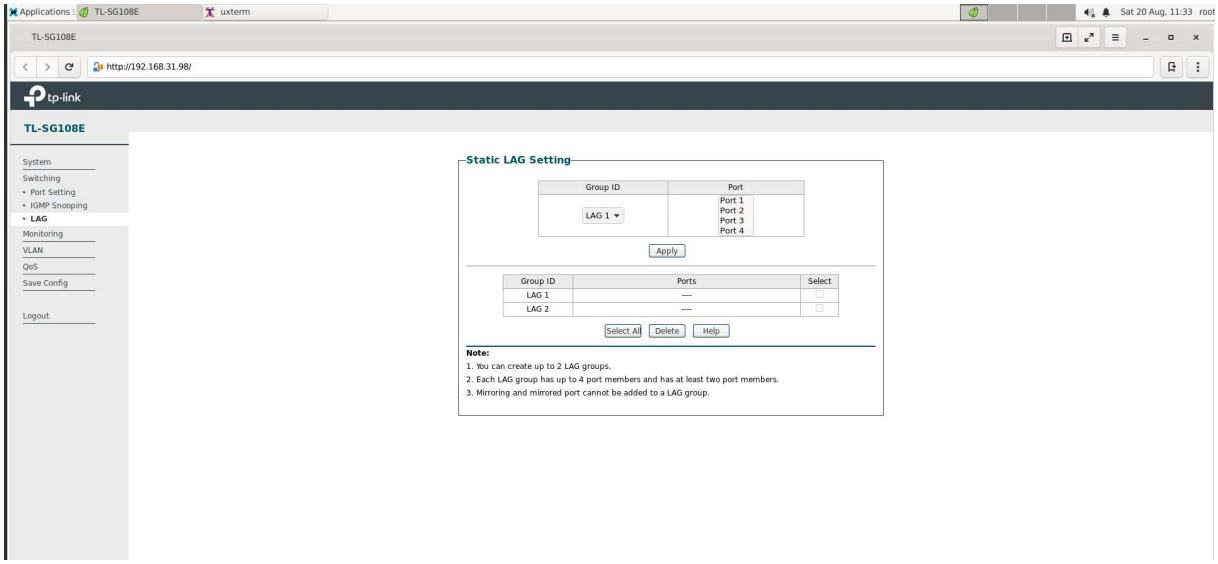
Fonte: Autoria própria (2022)

Figura 25 – *Reset* do switch



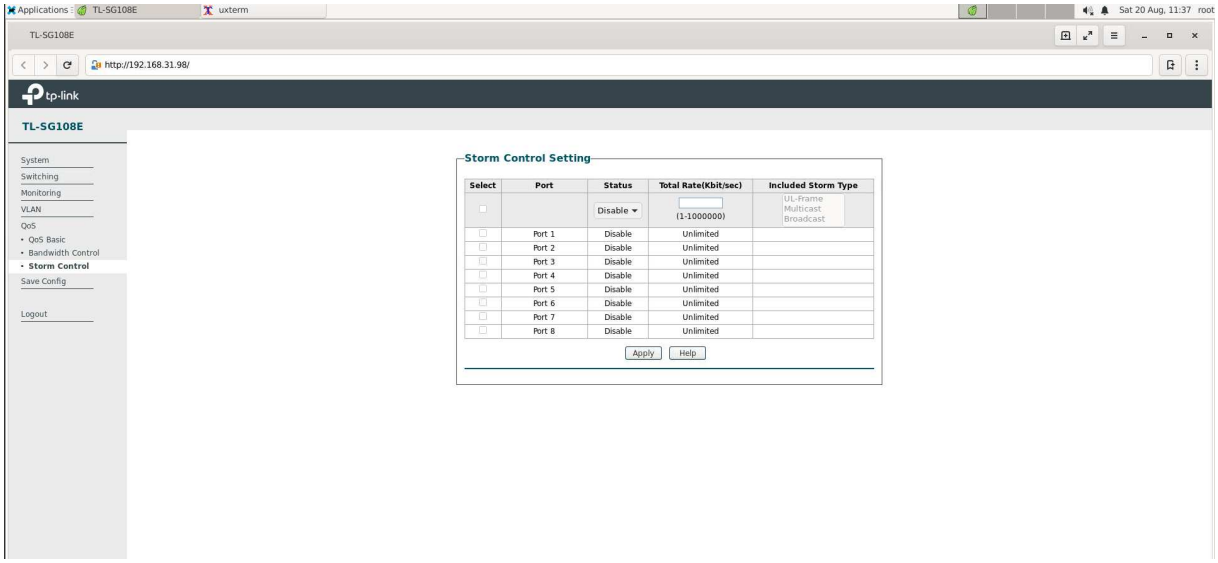
Fonte: Autoria própria (2022)

Figura 26 – Configuração de LAG estático



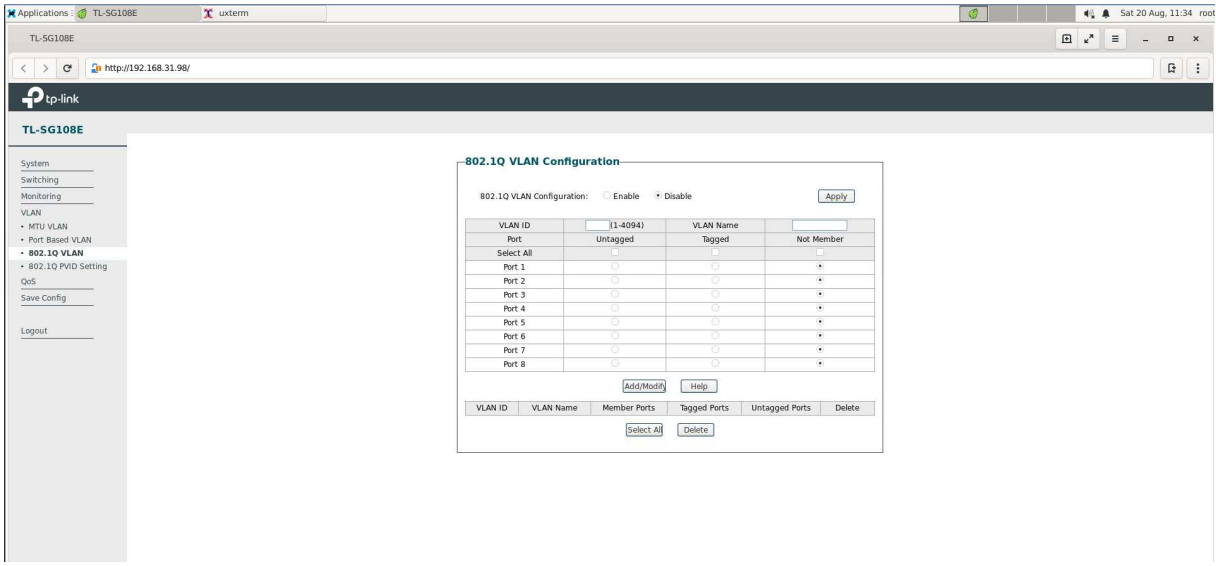
Fonte: Autoria própria (2022)

Figura 27 – Configuração de *storm control*



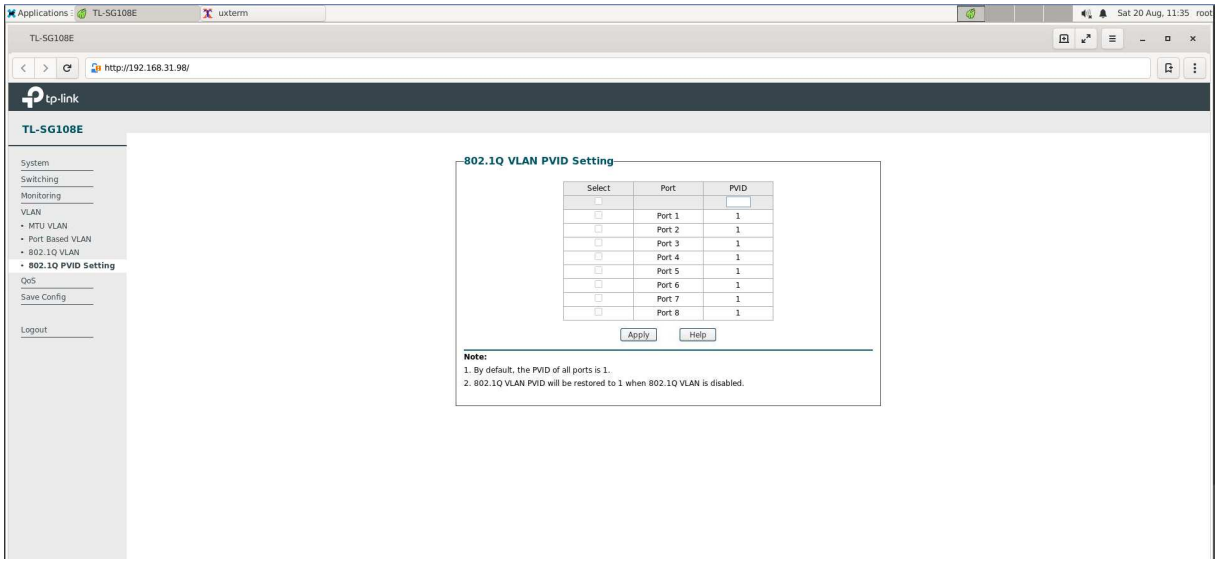
Fonte: Autoria própria (2022)

Figura 28 – Configuração de VLAN *wireless*



Fonte: Autoria própria (2022)

Figura 29 – Configuração de VLAN *wireless* com PVID



Fonte: Autoria própria (2022)

APÊNDICE B – Trechos do programa desenvolvido

Figura 30 – Método para criação de novo *switch* no banco de dados

```
async create({
                ipAddress,
                password,
                username
                }: CreateDeviceDto) {
   return await this.datasource.transaction(
   async (manager) => {
     const switchEntity = this
                           .devicessRepository
                           .create({
       username,
       ipAddress,
       password,
     })
     await manager
               .getRepository(Device)
               .save(switchEntity)
     const ports = Array<Port>()
     for (let i = 0; i < 8; i++) {
       ports.push(
         this.portsRepository.create({
           number: i + 1,
           deviceId: switchEntity.id,
           statusId: 1,
           speedId: 1,
           flowControlId: 1,
         }),
       )
     }
     await manager.getRepository(Port).save(ports)
     return switchEntity
   })
 }
```

Fonte: Autoria própria (2022)

Figura 31 – Método para remoção de *switch* do banco de dados

```
async remove(id: number) {
   return await this.datasource.transaction(
   async (manager) => {
     const switchEntity = await this
                                    .devicessRepository
                                    .findOneOrFail({
       where: { id },
       relations: { ports: true },
     })
     await manager.getRepository(Port)
                                    .remove(switchEntity.ports)
     return await manager
                         .getRepository(Device)
                         .delete(switchEntity)
   })
 }
```

Fonte: Autoria própria (2022)

Figura 32 – Método para atualização de um *switch* no banco de dados

```
async update(id: number, updateSwitchDto: UpdateDeviceDto) {
   await this.devicessRepository
                            .findOneOrFail({ where: { id } })
   return await this.devicessRepository.update(
     { id },
     {
       ipAddress: updateSwitchDto.ipAddress,
       password: updateSwitchDto.password,
       username: updateSwitchDto.username,
     },
   )
 }

```

Fonte: Autoria própria (2022)

Figura 33 – Método para atualização de uma port do *switch* no banco de dados

```
async update(updatePortDto: UpdatePortDto) {
    return await this.portsService.update(
      {
        number: updatePortDto.number,
        deviceId: updatePortDto.deviceId
      },
      {
        statusId: updatePortDto.statusId,
        speedId: updatePortDto.speedId,
        flowControlId: updatePortDto.flowControlId,
      },
    )
  }
```

Fonte: Autoria própria (2022)

Figura 34 – Método para acessar a interface *web* do *switch* e atualizar com as informações no banco de dados

```
@Cron(CronExpression.EVERY_10_MINUTES)
async updatePorts() {
    console.info(
    '[INFO]: Initalizing cron job to update ports for devices'
    )
    const crawler = await Crawler.init()
    try {
        if (!crawler) throw new Error('Crawler is null')
        const devices = await this.devicesRepository.find({
            relations: ['ports'],
            })
        if (!devices.length) {
            console.info(
            '[INFO]: No devices found, exiting...'
            )
            return
        }
        console.info(
        `[INFO]: Updating ${devices.length} devices`
        )
        for await (const device of devices) {
            await crawler.loginToSwitch(
            `http://${device.ipAddress}`,
            device.username,
            device.password,
            )
            const ports = device.ports.reverse()
            for await (const port of ports) {
                await crawler.changePortStatus(port)
            }
        }
    } catch (error) {
        console.error(
        `[ERROR]: Could not update ports ${error.message}`
        )
    } finally {
        await crawler.closeBrowser()
    }
}
```

Fonte: Autoria própria (2022)