UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS

RENATO FIORONI SIMÕES

SQUID CONFIGURATOR: AUTOMATIZANDO REDES

BAURU
2025

RENATO FIORONI SIMÕES

SQUID CONFIGURATOR: AUTOMATIZANDO REDES

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a conclusão do curso de Bacharelado de Sistemas de Informação da Faculdade de Ciências - UNESP Campus Bauru, sob orientação do Prof. Dr. Kelton Augusto Pontara Costa.

BAURU
2025

BANCA EXAMINADORA

Data: / /

_____

_____

_____

AGRADECIMENTOS

Agradeço à UNESP, ao curso de Sistemas de Informação e aos seus professores pela formação que recebi ao longo destes anos, a qual proporcionou a base necessária para o desenvolvimento deste trabalho e contribuiu decisivamente para minha evolução acadêmica e profissional.

Ao meu orientador, Kelton agradeço pela disponibilidade, pelo acompanhamento e, especialmente, pela direção dada no escopo inicial do projeto, em um momento em que eu não visualizava claramente quais soluções poderia propor com meus conhecimentos de programação e redes, os quais, naquela ocasião, se limitavam basicamente a Java e ao funcionamento de proxy.

Por fim, expresso minha profunda gratidão aos meus pais, Marcel e Cristiana, que me proveram todo o suporte material necessário durante minha formação e, principalmente, pelo amor, apoio incondicional e compreensão em todos os momentos, sendo fundamentais para que eu pudesse chegar até aqui.

RESUMO

Este trabalho visa desenvolver uma aplicação em Java para automação das configurações do servidor proxy Squid, com foco na otimização dos processos de gerenciamento de regras de acesso e controle de tráfego. O objetivo principal é facilitar e agilizar a edição do arquivo de configuração "*squid.conf*", tradicionalmente exigente em termos de conhecimento técnico e propenso a erros manuais. A metodologia utilizada inclui a implementação de uma solução intuitiva que permita a criação, modificação e exclusão de regras de acesso de forma automatizada, reduzindo o tempo e complexidade das operações e minimizando o risco a falhas na configuração. A aplicação desenvolvida foi testada em um ambiente virtualizado, onde foi avaliada a sua eficácia na gestão de um servidor proxy Squid em diferentes cenários de uso, enquanto o projeto também aborda a importância da automação em ambientes de rede, as características e funcionamento do Squid e a utilização do Java no gerenciamento do servidor.

Palavras-chave: Automação, Servidor Proxy, Squid, Java, Otimização de Processos.

ABSTRACT

This work aims to develop a Java application for automating the configurations of the Squid proxy server, focusing on optimizing the processes of managing access control rules and traffic management. The main objective is to facilitate and expedite the editing of the 'squid.conf' configuration file, which traditionally requires technical expertise and is prone to manual errors. The methodology includes the implementation of an intuitive solution that allows for the automated creation, modification, and deletion of access rules, reducing the time and complexity of operations while minimizing the risk of configuration errors. The developed application will be tested in a virtualized environment to assess its effectiveness in managing a Squid proxy server in various use scenarios. The project will also address the importance of automation in network environments, the characteristics and functioning of Squid, and the use of Java in server management.

***Keywords**: Automation, Proxy Server, Squid, Java, Process Optimization.*

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A administração e configuração de servidores proxy desempenham um papel central na gestão de redes corporativas, especialmente em ambientes que demandam controle rigoroso do acesso à internet, monitoramento de tráfego e adoção de políticas de segurança. Ferramentas como o Squid são amplamente utilizadas para realizar cache de conteúdo, aplicar filtros de acesso e otimizar a utilização da banda disponível. Entretanto, a configuração manual desses sistemas exige elevado nível de conhecimento técnico, precisão e atenção aos detalhes. Pequenos erros de sintaxe ou regras mal definidas podem comprometer o funcionamento adequado do servidor proxy e, por consequência, impactar diretamente o desempenho e a segurança da rede. Além disso, o cenário corporativo atual, marcado pela adoção contínua de novos protocolos e tecnologias, aumenta ainda mais a complexidade das tarefas realizadas pelos administradores (EDELMAN; LOWE; OSWALT, 2018, p.1).

Diante dessa realidade, torna-se pertinente o desenvolvimento de ferramentas que reduzam a carga operacional sobre administradores de rede, minimizem erros humanos e aumentem a padronização das configurações. A automação surge como alternativa eficaz para lidar com tarefas repetitivas e técnicas, permitindo que os profissionais se concentrem em atividades mais estratégicas.

O Squid, embora poderoso e amplamente utilizado, possui um arquivo de configuração extenso e sensível, que pode ser difícil de manipular em ambientes dinâmicos. Assim, uma aplicação que automatize esse processo representa uma contribuição relevante, especialmente para organizações que buscam agilidade, flexibilidade e segurança na gestão de suas políticas de acesso à internet. A motivação para o desenvolvimento desta solução decorre da necessidade prática de facilitar o trabalho cotidiano de administradores de rede. A edição de regras no *squid.conf* é repetitiva, sujeita a erros e demanda tempo significativo. Além disso, muitos profissionais iniciantes enfrentam dificuldades com a sintaxe do Squid, o que pode atrasar a implementação de novas políticas ou correções no ambiente. A possibilidade de automatizar esse processo em uma interface amigável, que traduza operações complexas em ações simples, representa um ganho direto em produtividade, confiabilidade e qualidade operacional.

1.2 Objetivo Geral e Objetivos Específicos

O objetivo geral do projeto é desenvolver uma aplicação em Java capaz de automatizar a configuração do servidor proxy Squid, permitindo a criação, modificação e remoção de regras de forma simples, segura e eficiente. Como objetivos específicos:

- Analisar o funcionamento do Squid e sua estrutura de configuração no arquivo *squid.conf*;
- Desenvolver uma interface gráfica intuitiva, capaz de abstrair a complexidade das regras;
- Implementar uma forma de autenticação para acesso ao sistema;
- Implementar funcionalidades que criem, editem e removam ACLs, políticas de acesso, regras de cache e controle de banda;
- Garantir que a aplicação valide as regras geradas antes de aplicá-las ao arquivo de configuração;
- Oferecer a capacidade de se realizar comandos para iniciar, reiniciar, parar e recarregar o serviço do Squid;
- Testar a solução em ambientes virtualizados, assegurando sua funcionalidade, robustez e confiabilidade;
- Avaliar a redução de erros e o ganho de produtividade proporcionados pela automação.

2 AUTOMAÇÃO EM AMBIENTES DE REDE

2.1 Detalhamento do Problema

No contexto de redes empresariais, a configuração e o gerenciamento de dispositivos de rede, como switches, roteadores e servidores, muitas vezes seguem processos manuais e altamente suscetíveis a erros. Cada dispositivo pode ser visto como um “grão de areia único”, com configurações personalizadas que os tornam complexos de manter e atualizar. Essa falta de padronização torna a rede não apenas complexa, mas também frágil, uma vez que pequenas mudanças podem causar interrupções significativas no ambiente operacional.

Com a situação explicitada, configurações manuais se tornam exigentes em tempo e esforço dos administradores de rede. Cada nova implementação, como a configuração de uma VLAN, a criação de regras de firewall, a adoção ou ajuste de políticas de proxy, ou apenas a resolução de problemas de conectividade pode levar horas e até dias, dependendo da complexidade da rede e do número de dispositivos envolvidos. Ainda assim, a equipe de TI, enquanto é constantemente desafiada a manter o ritmo da resolução de demandas empresariais cada vez mais rápidas, também precisa enfrentar os contratempos e minimizar os riscos. Instituições que dependem de grandes infraestruturas de rede, como provedores de serviços, instituições financeiras, empresas de tecnologia e órgãos governamentais, são diretamente impactadas por essa situação.

A falta de automação nas redes pode levar a tempo de inatividade prolongado, perda de produtividade e, em casos mais graves, à falha no fornecimento de serviços essenciais, além de brechas na segurança. Edelman, Lowe e Oswalt (2018, p.19, tradução nossa) ressaltam que "o uso da automação de rede comprovada e testada para fazer alterações ajuda a obter um comportamento mais previsível do que fazer alterações manualmente e dá à equipe uma chance melhor de alcançar resultados determinísticos", o que reforça a importância de adotar soluções automatizadas para minimizar riscos e melhorar a eficiência. A crescente demanda por soluções de rede que sejam rápidas, confiáveis e escaláveis coloca pressão sobre as equipes de TI para adotar práticas mais eficientes. Em um mundo onde as operações estão cada

vez mais interligadas por redes, a capacidade de gerir essas redes de forma eficaz é crucial para o sucesso de qualquer organização.

2.2 Vantagens da Automação e Objetivo da Proposta

A automação de redes oferece uma série de vantagens que tornam o processo de gestão mais eficiente e confiável:

- Velocidade e Agilidade: com a automação, as configurações podem ser implementadas de maneira mais rápida, permitindo que as redes se adaptem rapidamente às necessidades do negócio. Mudanças que antes exigiam horas de trabalho manual podem ser realizadas em minutos;

- Arquiteturas Simplificadas: ao automatizar o processo de configuração, as redes ganham unificação e se tornam mais fáceis de gerenciar. "A simplificação da arquitetura resulta em uma rede que é mais fácil de manter e automatizar, com menos extensões proprietárias ao longo de sua estrutura" (EDELMAN; LOWE; OSWALT, 2018, p.18, tradução nossa);

- Resultados Determinísticos: a automação reduz drasticamente a possibilidade de erros humanos, proporcionando resultados previsíveis. Com isso, os profissionais de TI podem ter mais confiança de que as mudanças feitas terão o efeito desejado sem causar problemas imprevistos;

- Redução de Custos: ao reduzir a necessidade de intervenção manual, as empresas podem diminuir os custos operacionais, tanto em termos de horas de trabalho quanto na correção de erros que podem ocorrer durante mudanças de configuração;

- Escalabilidade: à medida que a rede cresce, torna-se inviável manter processos manuais. A automação permite que a infraestrutura de rede cresça de forma mais organizada e segura, sem aumentar proporcionalmente a carga de trabalho da equipe de TI;

Apesar de suas vantagens, a automação de redes encontra obstáculos, sendo um dos principais a falta de padronização nas plataformas de rede. Cada fabricante oferece APIs e ferramentas diferentes, o que torna a implementação de uma solução automatizada universal mais complexa. Além disso, a automação de processos manuais já consolidados requer uma compreensão detalhada dos fluxos de trabalho existentes, automatizar processos sem entender suas nuances pode introduzir novos problemas em vez de resolvê-los.

Com isso em vista, nossa proposta busca contribuir com o ambiente de rede empresarial ao automatizar um processo específico, o da configuração do popular servidor proxy Squid, facilitando a criação e o gerenciamento de regras diretamente no arquivo "*squid.conf*", por meio de uma aplicação em Java. A aplicação terá uma interface gráfica amigável e será executada diretamente no servidor, garantindo uma configuração simplificada e eficiente.

2.3 Solução Existente

Há ferramentas já desenvolvidas capazes de realizar o gerenciamento do Squid, driblando sua complexidade, a mais documentada e utilizada é o Webmin.

2.3.1 Webmin

O Webmin é uma interface baseada em web que tem por principal objetivo administrar sistemas Unix, podendo configurar e monitorar usuários, cotas de disco, serviços, arquivos, entre outros aspectos do Sistema Operacional. A Figura 1 apresenta o painel principal da plataforma.

Figura 1 – Painel Principal do Webmin

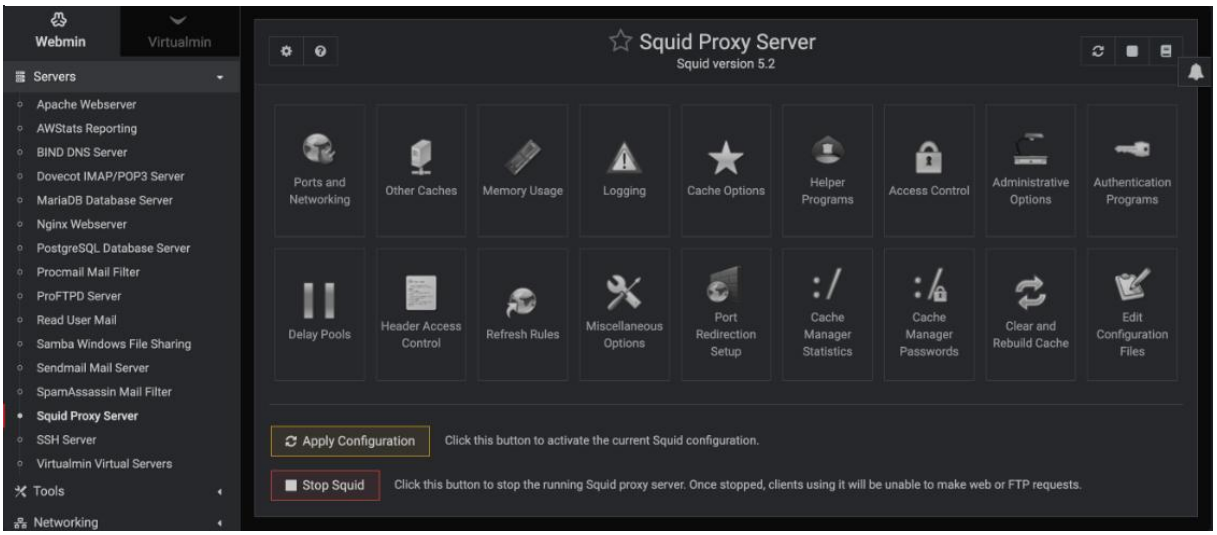
Fonte: WEBMIN (2024)

A plataforma possui diversos módulos que podem ser instalados separadamente, dentre eles, um módulo de configuração do Squid.

> Se você quiser instalar ou configurar o Squid de dentro do Webmin, você precisará usar o módulo Squid Proxy Server, encontrado na categoria *Servers*. Quando seu ícone for clicado, a página mostrada na captura de tela abaixo aparecerá, assumindo que o Squid esteja instalado e configurado corretamente. Como você pode ver, a página principal consiste apenas em uma tabela de ícones, cada um dos quais pode ser clicado para abrir um formulário para editar as configurações naquela categoria. (CAMERON; COOPER, 2023, tradução nossa)

O menu de configuração do Webmin pode ser observado na Figura 2.

Figura 2 – Menu do Módulo de Configuração do Squid no Webmin



Fonte: WEBMIN (2024)

O Webmin é uma plataforma robusta e com muitos pontos positivos, dentre eles: possui uma interface amigável, possibilitando que usuários com pouca experiência possam utilizá-lo sem grandes dificuldades, e dá suporte a uma ampla gama de configurações e serviços do Unix, o que o torna uma ferramenta versátil e completa para administração de sistemas. Por outro lado, também é possível constatar pontos negativos da ferramenta, especialmente quando se olha a ela como uma solução para a configuração do servidor proxy Squid: requer instalação e configuração de um servidor web, o que adiciona camadas de complexidade, depende de uma interface gráfica, impossibilitando seu uso em ambientes em que não há interface e não é uma aplicação específica para a configuração do Squid, sendo mais complexa e menos otimizada para aqueles que apenas desejam se utilizar dessa função.

2.3.2 Vantagens da Solução Proposta

Muitas das vantagens da aplicação proposta neste trabalho em relação ao Webmin estão diretamente relacionadas a um aspecto da plataforma já existente: seu uso mais amplo. Com o fato de o Webmin não ser voltado apenas à edição das regras e aspectos do Squid, nossa solução ganha vantagem ao: ser mais leve e fácil de usar, dotar de uma interface mais direta e de uso otimizado e não possuir funcionalidades desnecessárias, tendo foco nas necessidades de automação específicas do servidor proxy. Além disso, nossa solução é menos dependente de recursos externos, funcionando como um software de desktop que é independente de um servidor web adicional.

Antes de adentrarmos nos aspectos em si da aplicação a ser desenvolvida, onde serão apresentados suas funcionalidades, arquitetura, diagramas, detalhes de implementação, métodos de validação e resultados, é importante que seja abordado de forma aprofundada o servidor proxy Squid. Assim, será possível que se entenda o funcionamento de um servidor proxy em geral, as características específicas do Squid e como foi realizada sua configuração neste trabalho.

3 SERVIDOR PROXY SQUID

3.1 Funcionamento de um Servidor Proxy

Um servidor proxy atua como intermediário entre um cliente e o servidor de destino ao qual o cliente deseja se conectar, recebendo solicitações e respostas de um lado e as transmitindo ao outro (atuação que é ilustrada na Figura 16). O proxy facilita essa comunicação sem alterar as solicitações ou respostas, dando a impressão de que o cliente se comunica diretamente com o servidor de destino. Assim, ele “toma posse” da conexão do cliente e se apresenta como cliente ao servidor de destino, o que permite um controle mais eficiente das comunicações (SAINI, 2011, p. 27).

Em configurações mais avançadas, o proxy pode analisar e filtrar solicitações e respostas de acordo com regras predefinidas, que consideram critérios como o endereço IP do cliente ou do servidor, o tipo de conteúdo solicitado e o protocolo utilizado. A técnica de armazenamento em cache também é uma funcionalidade central dos proxies, pois permite que eles guardem localmente respostas para solicitações frequentes, economizando largura de banda e reduzindo o tempo de carregamento das páginas. Essas características contribuem para que uma série de medidas de rede elaboradas por empresas possam ser postas em prática, como: implantar políticas de acesso à rede, monitorar e registrar o tráfego de usuários, proteger a privacidade dos usuários na internet e muitas outras.

3.2 Introdução ao Squid

A primeira versão do Squid começou a ser desenvolvida como parte do Projeto Harvest, que tinha como um de seus objetivos o estudo e criação de tecnologias de cache. O projeto teve seu início na Universidade do Colorado, em Boulder e foi aprofundado na Universidade da California, em San Diego.

Tendo seu primeiro lançamento em julho de 1996 e atualmente em sua versão 6.12 (outubro de 2024), o Squid é um servidor proxy que suporta os protocolos HTTP, HTTPS, FTP, e muitos outros. Possuindo todas as características de um servidor proxy como explicitado no capítulo anterior, ele se destaca por ser um dos mais utilizados, e consequentemente, possuir ampla documentação, guias, exemplos de uso e uma

comunidade ativa. Além disso, mesmo sendo de código aberto e gratuito, ele é conhecido por ter grande flexibilidade de configuração, armazenamento em cache eficiente, ter ótimo desempenho e ser compatível com a maioria dos sistemas operacionais disponíveis no mercado (SQUID CACHE, 2024). A logo oficial do squid é apresentada na Figura 3.

Figura 3 – Logo do Squid Proxy Server

Fonte: SQUID CACHE (2024)

Todos os pontos de destaque do Squid em relação a outros servidores de proxy também foram motivos para a escolha de sua utilização neste trabalho, especialmente sua ampla utilização, documentação e natureza de código aberto.

3.3 Funcionamento e Configuração Básica do Squid

O funcionamento e configuração do Squid são centralizados no arquivo *squid.conf*, o qual recebe todos os tipos de regras que o servidor cobre na forma de comandos, que possuem uma sintaxe específica; o principal tipo desses comandos são as chamadas ACLs (*Access Control Lists*). Após a reinicialização do servidor, todas as regras adicionadas entrarão e continuarão em vigor até que sejam comentadas ou excluídas do arquivo e o Squid seja reiniciado novamente.

A seguir, serão colocados em sequência os procedimentos — realizados neste projeto — para instalação e configuração inicial do Squid, que abrangem tanto comandos de terminal do sistema operacional Ubuntu-Linux quanto de configuração do *squid.conf*, com sua sintaxe própria:

1. Instalação do Squid: suas diretrizes já vêm como parte do pacote de instalação do Ubuntu Desktop, portanto é necessário apenas a realização do comando `sudo apt install squid`;

2. Proteção do arquivo original: assim como recomendado em UBUNTU SERVER (2024), é mais seguro que se crie uma cópia do arquivo de configuração original sem alterações realizadas e que se o bloqueie de receber escritas, respectivamente com os comandos `sudo cp /etc/squid/squid.conf /etc/squid/squid.conf.original` e `sudo chmod a-w /etc/squid/squid.conf.original`;

3. Acesso ao *squid.conf*: para acessar o arquivo de configuração no modo de edição, realiza-se o comando `sudo nano /etc/squid/squid.conf`;

4. Configuração de porta: o Squid por padrão responde à porta 3128 e isso pode ser alterado, o que não foi necessário neste projeto, então no arquivo adiciona-se `http_port 3128`;

5. ACL inicial: por padrão o Squid deverá recusar qualquer requisição de clientes, portanto, é necessário colocar como uma primeira regra a permissão do acesso HTTP a partir de uma rede específica (IPs num determinado intervalo). Para isso, cria-se uma ACL “rede local” que representa essa rede e se permite o acesso a partir dela às requisições HTTP com os comandos `acl localnet src 192.168.56.0/24` e `http_acess allow localnet`;

6. Definição de portas de destino seguras: é recomendável que se utilize das ACLs para definir as portas de destino consideradas seguras e se bloqueie o acesso àquelas que não são, para isso, sabendo — por exemplo — que a porta para o HTTP padrão é a 80 e para HTTPS é a 443, utiliza-se os comandos `acl Safe_ports port 80`, `acl Safe_ports port 443` e `http_acess deny !Safe_ports`;

7. Bloqueando acessos não autorizados: para bloquear qualquer acesso que não seja autorizado, na última linha do arquivo adiciona-se `http_access deny all`;

8. Reiniciando o Squid: por fim, para que as regras adicionadas no *squid.conf* entrem em vigor, basta reiniciar o Squid, realizando no terminal o comando `sudo systemctl restart squid`;

9. Monitoramento: para se verificar o arquivo de configuração *squid.conf* em busca de erros, pode-se utilizar do comando `sudo squid -k parse`, já para apenas verificar se o Squid está ativo, basta realizar `sudo systemctl status squid`.

Feitos esses passos, o Squid está configurado e funcionando em sua forma mais básica, estando pronto para receber configurações mais avançadas. É importante salientar que, a comunicação do servidor Squid com o cliente que foi preparada nesse projeto, assim como o procedimento de a estabelecer, será adentrada no capítulo 4, na seção “Procedimento de Validação”.

3.4 Configuração Avançada do Squid

Esta seção terá por objetivo descrever e exemplificar as principais funcionalidades do Squid e como elas são configuradas diretamente no servidor. Tais funcionalidades também serão aquelas contempladas pela automação promovida na solução proposta, porém nesse momento serão abordadas sem o contexto da automação e apenas em sua forma “crua”.

3.4.1 Access Control Lists (ACLs)

Como já comentado, as ACLs são a parte central do *squid.conf*. Elas levam em conta diferentes aspectos específicos de requisições feitas por clientes, como endereços IP, endereços URL, método da requisição e o número da porta dos servidores de origem e destino; assim, é possível estabelecer políticas de acesso

diferentes de acordo com esses aspectos. Após criadas, as ACLs são combinadas com uma série de diretivas, no objetivo de permitir ou não o acesso à rede com base nas condições estabelecidas (WESSELS, 2004, p. 111). Os tipos mais comuns de ACL presentes no Squid são:

- src: baseadas em endereços IP de origem. Exemplo:

```
acl local_network src 192.168.56.0/24;
```

- dst/dstdomain: baseadas em endereços IP ou domínio de destino. Exemplo:

```
acl blocked_site dstdomain facebook.com;
```

- port: controlam as portas de destino. Exemplo:

```
acl Safe_ports port 80;
```

- time: controlam o acesso baseado em horários estabelecidos. Exemplo:

```
acl working_hours time MTWHF 08:00-18:00;
```

- url_regex: definem filtros baseados em expressões ou palavras-chave. Exemplo:

```
acl blocked_words url_regex -i instagram.
```

Cada ACL pode ter um ou mais valores associados a ela (por exemplo: acl local_network src pode possuir vários endereços IP) e ser associada a uma diretiva, a qual pode ser acompanhada de uma ação *allow* ou *deny* . São os principais tipos de diretiva:

- http_access: controla quem pode fazer requisições HTTP. Exemplo:

```
http_access allow local_network
http_access deny bloqued
```

- http_reply_access: controla quem pode ver as respostas HTTP dos servidores (o cliente pode até realizar a requisição, mas não terá acesso a resposta). Exemplo:

```
http_reply_access deny pornografia
```

```
http_reply_access allow all
```

- url_rewrite_access: controla com quem o Squid pode reescrever URLs. Exemplo:

```
url_rewrite_access allow local_network
url_rewrite_access deny all
```

- delay_access: diz quem será afetado pelas regras de limite de largura de banda. Exemplo:

```
delay_access pool1 allow usuarios_normais
delay_access pool1 deny admin
```

- cache: controla o que pode ser armazenado em cache. Exemplo:

```
cache allow arquivos_estaticos
cache deny conteudo_dinamico
```

- access_log: define qual tráfego deve ser registrado em log e para qual arquivo. Exemplo:

```
access_log /var/log/squid/acesso.log squid
access_log none conteudo_privado
```

3.4.2 Gerenciamento de Cache

O Squid é amplamente utilizado por suas operações envolvendo cache. As regras de cache configuradas no servidor definem como e quando os dados devem ser armazenados e reutilizados. Assim, ele permite que o uso de banda seja otimizado e o tempo de carregamento de páginas para os usuários diminua, o que o torna uma ferramenta essencial em ambientes onde a eficiência de rede é um aspecto crucial.

O cache no Squid funciona de forma que, temporariamente, armazena conteúdos frequentemente acessados pelo usuário, o que permite que em requisições posteriores ele ofereça esse conteúdo diretamente do cache, economizando largura de banda e tempo. Esse armazenamento temporário é configurável para especificar quais tipos de conteúdo devem ou não ser armazenados, por quanto tempo e sob que

condições podem ser servidos em cache (WESSELS, 2004, p. 156).

Os parâmetros configuráveis de cache no Squid são os seguintes:

- Cache_mem: define a quantidade de memória RAM alocada para o cache de objetos recentemente acessados. Exemplo:

```
cache_mem 256 MB;
```

- Cache_dir: define o diretório do cache em disco, o tamanho total de armazenamento e os parâmetros da quantidade de subdiretórios no cache, respectivamente. Exemplo:

```
cache_dir ufs /var/spool/squid 10240 16 256;
```

- Maximum_object_size: define o tamanho máximo de cada objeto que pode ser amazenado em cache. Exemplo:

```
maximum_objetct_size 4096 KB;
```

- Minimum_object_size: define o tamanho mínimo de cada objeto que pode ser armazenado em cache. Exemplo:

```
minimum_object_size 1 KB;
```

- Refresh_pattern: configura políticas de atualização e expiração dos conteúdos em cache com base em parâmetros específicos; os demonstrados no exemplo são: tipo de conteúdo (.jpg), tempo mínimo de validade em cache (em minutos), percentual de extensão de tempo caso o objeto seja modificado e tempo máximo de validade em cache (também em minutos). Exemplo:

```
refresh_pattern -i \.jpg$ 1440 20% 10080;
```

- Cache_swap_low e cache_swap_high: controlam a limpeza de cache a partir de uma porcentagem de ocupação estabelecida. Quando ele atinge a porcentagem *low*, o cache começa a ser liberado de forma controlada e gradual, já quando atinge a porcentagem *high*, essa limpeza é intensificada.

Exemplos:
```
cache_swap_low 85
cache_swap_high 95.
```

3.4.3 Controle de Largura de Banda

Assim como o gerenciamento de cache, o controle de largura de banda no Squid tem por objetivo tornar o uso da rede mais eficiente, porém agora ao limitar e gerenciar o seu consumo, regulando a quantidade de dados que podem ser transmitidos num determinado intervalo de tempo. É útil especialmente em ambientes onde a conexão à internet possui capacidade limitada e se deseja garantir uma experiência equilibrada para cada tipo de usuário.

As regras de limites de rede podem ser aplicadas em diferentes níveis, como por usuário, IP, tipo de conteúdo ou horário, e o servidor utiliza de ACLs e diretivas específicas em conjunto para criar essas regras, diretivas que consistem em:

- Delay_pools: define a quantidade de *pools* (grupos) de controle de largura de banda que devem ser gerenciados. Cada *pool* poderá ser configurado com regras próprias. Exemplo:
  ```
  delay_pools 1;
  ```

- Delay_class: determina a classe de um *pool* criado. Assim como é abordado por Wessels (2004, p. 634), existem três tipos de classe:

  1. De controle global (para todo tráfego);
  2. De controle por rede (IP de origem ou destino);
  3. De controle por rede e usuário (a mais detalhada).

  Exemplo onde a *pool* 1 é declarada do tipo 2:
  ```
  delay_class 1 2;
  ```

- Delay_access: define quais ACLs estão sujeitas às regras de um pool específico. Com essa diretiva é possível definir quais usuários ou redes estarão

sob limitação de largura de banda. Exemplo:

```
acl local_network src 192.168.56.0/24
delay_access 1 allow local_network;
```

- Delay_parameters: configura os limites reais de largura de banda de um *pool* específico. Os valores serão expressos em bytes por segundo (download/upload) e divididos em até três partes dependendo do tipo de classe do *pool* (de acordo com as ACLs de usuários e IPs atribuídos). Exemplo para classe do tipo 1:

```
delay_pools 1
delay_class 1 1
delay_parameters 1 512000/512000.
```

3.4.4 Criação e Autenticação de Usuários

Por fim está uma funcionalidade essencial e muito utilizada em servidores proxy especialmente por grandes instituições e companhias, a criação e autenticação de usuários. Com ela é possível que apenas pessoas autorizadas, com nome de usuário e senha criados, possam ter acesso à rede.

Existem formas diferentes para que o Squid possa autenticar usuários, aqui será exposta a mais comum e básica delas. Primeiramente, é necessário que sejam realizados os seguintes passos no terminal do servidor, fora do arquivo de configuração:

- Instalar o utilitário de autenticação: o Squid fara uso do utilitário htpasswd, presente no pacote apache2-utils; para o instalar usa-se o comando `sudo apt-get install apache2-utils`;

- Criar um arquivo de senhas: utilizando do htpasswd, cria-se um arquivo de senhas e um primeiro nome de usuário com o comando `sudo htpasswd -c /etc/squid/passwd nome_de_usuario`, em seguida, será solicitada a senha e confirmação de senha do usuário;

- Adicionar novos usuários: para adicionar mais usuários basta utilizar do mesmo comando, porém sem o parâmetro “-c”, que cria o arquivo; a senha e confirmação continuarão sendo solicitadas após a criação do usuário com `sudo htpasswd /etc/squid/passwd outro_usuario`.

Para configurar o Squid de forma que permita o acesso apenas para usuários autenticados, adiciona-se as seguintes linhas de diretrizes e ACLs no *squid.conf*:

```
auth_param basic program /usr/lib/squid/basic_ncsa_auth /etc/squid/passwd
auth_param basic realm Squid proxy-caching web server
auth_param basic credentialsttl 2 hours
auth_param basic casesensitive on
acl autenticados proxy_auth REQUIRED
http_access allow autenticados
```

Após aplicadas as regras, quando o cliente entrar num navegador e tentar acessar qualquer site, será solicitada pelo proxy a autenticação via nome de usuário e senha, caso não sejam fornecidos, o acesso à página será negado.

3.5 Processo de Automação

Parte principal das funcionalidades apresentadas estará presente na aplicação desenvolvida, porém para serem aplicadas no Squid não necessitarão de intervenção manual direta no arquivo, como todos os comandos explicitados sugeririam. Esses comandos e diretrizes do arquivo *squid.conf* serão escritos de forma automática através das interações do usuário com a interface gráfica e a própria aplicação terá a atribuição de demonstrar ao usuário as regras que estão ativas, o status do servidor e garantir a conformidade, organização e funcionamento correto do arquivo *squid.conf.*

4 SOLUÇÃO IMPLEMENTADA

Tendo-se conhecimento daquilo que envolve aplicação e sua finalidade, pode-se afirmar que de acordo com a categorização de tipos de automação de redes estabelecida por Edelman, Lowe e Oswalt (2018, p. 25), essa solução se encaixa na categoria de Gerenciamento de Configuração (a mais comum), pois ao criar uma aplicação Java para editar o arquivo *squid.conf*, o gerenciamento das configurações se torna mais eficiente e menos propenso a erros.

O projeto carrega quatro exigências principais, sendo elas: automação de configuração do squi.conf (realizar suas configurações sem necessidade de intervenção manual no arquivo), validação de configuração (validar as novas regras antes de aplica-las, no objetivo de evitar conflitos), aplicação local de configurações (aplicar as mudanças diretamente no servidor onde a aplicação está sendo executada) e interface gráfica (dotar de uma interface que proporcione uma interação intuitiva e completa para a criação de regras do Squid).

4.1 Fluxo de Utilização

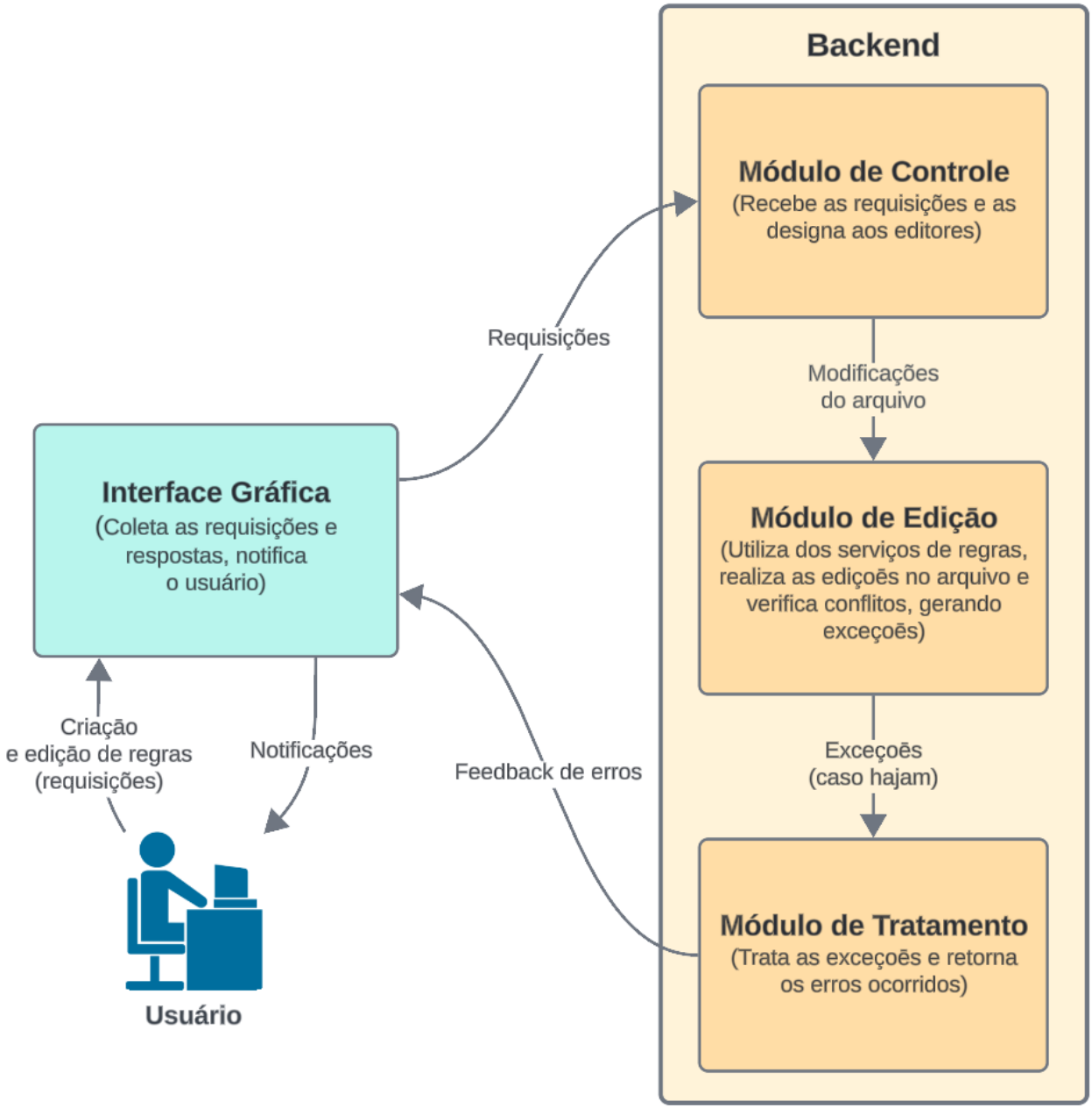
O fluxo de utilização será apresentado através da utilização da interface gráfica desenvolvida, porém é importante salientar que o backend é desacoplado do frontend, então os endpoints do sistema ficam disponíveis para requisições serem recebidas de outras fontes além da interface implementada.

Após realizar o *login*, o usuário seleciona que tipos de configurações de regras de proxy deseja realizar e será guiado para uma tela específica de cada tipo de regra. Após as configurações realizadas (adicionar novas regras, editar regras, adicionar diretivas, valores, etc.) ele as salva, momento em que o programa verifica o arquivo em busca de conflitos; caso algum conflito for detectado, o usuário deve ser notificado e as operações solicitadas descartadas, caso não sejam constatados conflitos, elas serão aplicadas no arquivo. As modificações entrarão em vigor no Squid quando o usuário o recarregar na página de servidor. O usuário também poderá visualizar a qualquer momento, sem interação direta com o arquivo de configuração, quais regras estão aplicadas com êxito (estando ativas) e o status do Squid naquele momento,

assim como poderá a qualquer momento interromper seu funcionamento ou o iniciar novamente.

O fluxo de utilização descrito, ao olhar de organização da aplicação, pode ser resumido nos módulos que serão a seguir apresentados num diagrama de blocos (Figura 4).

Figura 4 – Diagrama em Blocos da Aplicação



Fonte: Elaboração própria.

4.2 Tecnologias Utilizadas

Nesta seção serão abordadas as tecnologias utilizadas no desenvolvimento do projeto, assim com os motivos que embasaram suas escolhas.

4.2.1 Frontend

A interface gráfica foi desenvolvida utilizando React, uma biblioteca JavaScript amplamente adotada para construção de interfaces modernas, dinâmicas e componentizadas. O React ofereceu alta flexibilidade e permitiu criar telas responsivas com uso integrado de HTML, CSS e JavaScript, garantindo uma experiência visual mais fluida e atual. Além disso, seu ecossistema rico (incluindo ferramentas como Vite, React Router e diversas bibliotecas de UI) tornou o desenvolvimento mais seguro e ágil. O uso de componentes reutilizáveis também facilitou a manutenção e a expansão futura do sistema.

Outras alternativas, como JavaFX ou até mesmo o uso direto de HTML/CSS sem frameworks foram consideradas, porém acabaram descartadas devido às suas limitações em termos de escalabilidade, organização e modernidade. Dessa forma, o React se mostrou a solução mais adequada para atender às necessidades do projeto, oferecendo uma interface mais agradável, modular e alinhada com as tecnologias predominantes no mercado.

4.2.2 Backend

O backend do sistema foi desenvolvido utilizando Java, Spring Boot e Maven, um conjunto de tecnologias escolhidas pela maturidade, estabilidade e ampla adoção no desenvolvimento de aplicações corporativas. O Spring Boot oferece uma estrutura robusta para criação de APIs REST e simplifica significativamente a configuração do projeto, enquanto o Maven é responsável pelo gerenciamento automatizado de dependências e pelo ciclo de build, garantindo padronização, organização e facilidade

na manutenção do código. Esse ecossistema fornece uma base segura, escalável e produtiva, totalmente adequada ao propósito do sistema: interpretar, manipular e aplicar configurações do Squid de forma automatizada.

4.3 Funcionalidades

A seleção de funcionalidades do Squid que foram implementadas nessa primeira versão do projeto foi baseada em sua criticidade no controle e otimização do tráfego de rede, além de sua maior viabilidade para serem configuradas de maneira programática.

As funcionalidades de adição de regras foram previamente abordadas no capítulo “Configuração Avançada do Squid”, assim, nesse momento, serão apresentadas sua operação no contexto da aplicação e seus desafios de implementação:

- Controle de Acesso (ACLs): a aplicação fornece a capacidade ao usuário de adicionar, remover e editar regras ACL de forma intuitiva. Além disso, a aplicação permite visualizar facilmente uma lista das regras que já estejam em vigor, incluindo seus valores e diretivas. O principal desafio foi abranger todos os principais tipos de ACL, garantir que as regras não entram em conflito entre si e que as configurações inválidas não sejam aplicadas;

- Adição de Regras de Cache: a aplicação inclui uma aba para configurar parâmetros importantes como tempo de retenção de objetos no cache, tamanho máximo dos objetos e quantidade total de espaço disponível para cache. Novamente, o principal desafio foi garantir que regras conflitantes não fossem adicionadas no arquivo;

- Adição de Regras de Largura de Banda: foi incluída na aplicação uma aba para configurar limites de largura de banda. Agrupar os parâmetros de cada

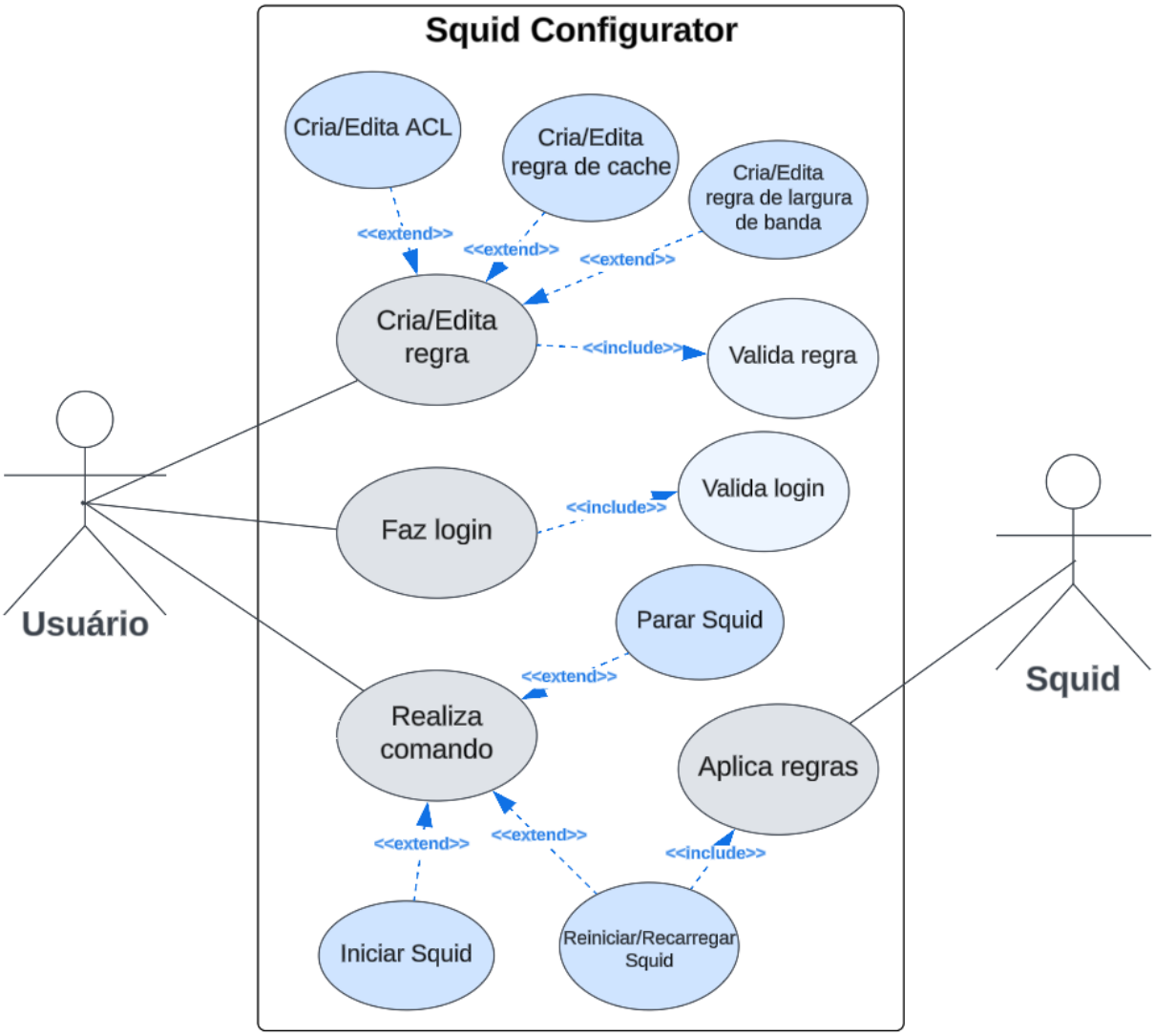
respectivo pool e garantir que a sequência correta de adição de diretivas fosse seguida foram os principais dificultadores;

- Controle do servidor: foi criada uma aba onde o usuário pode gerenciar o funcionamento do Squid, podendo-o: recarregar (para que as alterações do arquivo entrem em vigor), iniciar, reiniciar e parar. Ele também pode visualizar o conteúdo atual do *squid.conf* “cru” e adicionar/retirar o bloqueio de acesso geral (http_access deny all);

- Autenticação de usuário: a autenticação adotada no sistema utiliza um mecanismo simples baseado em verificação direta de credenciais no backend. O usuário pode apenas acessar as outras rotas do front (protegidas) após realizado o login na página inicial.

Os Todas essas funcionalidades podem ser observadas no diagrama de casos de uso apresentado a seguir (Figura 5).

Figura 5 – Diagrama de Casos de Uso da Aplicação



Fonte: Elaboração própria.

4.4 Detalhes de Implementação

Este capítulo apresenta os principais aspectos da implementação do sistema, descrevendo como as tecnologias escolhidas foram estruturadas e integradas no decorrer do desenvolvimento. Após passar pela interface, definir a arquitetura e os componentes fundamentais, serão detalhadas as responsabilidades centrais de cada camada (*Controller*, *Service* e *Editor*) evidenciando como elas colaboram para garantir

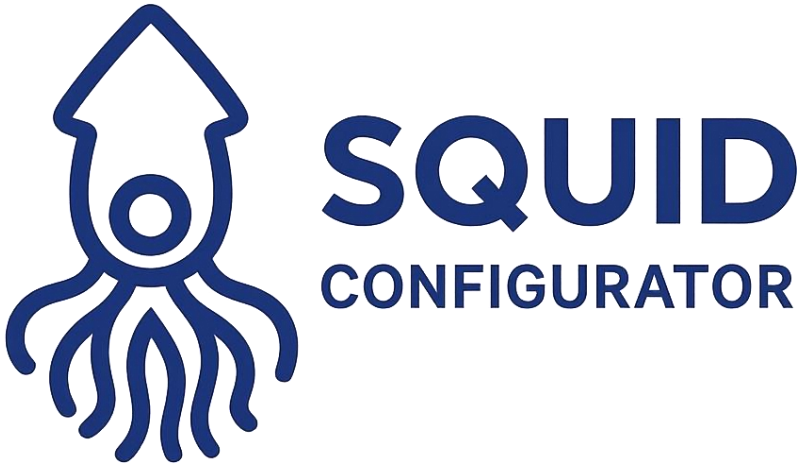
o funcionamento correto do sistema. O objetivo não é explorar exaustivamente cada trecho de código, mas destacar os elementos essenciais que tornam possível a comunicação entre o frontend e o backend, o processamento das regras do Squid e a execução dos comandos no servidor. Dessa forma, o capítulo fornece uma visão clara e prática de como o sistema foi construído internamente, evidenciando as decisões técnicas que sustentam seu funcionamento.

4.4.1 Frontend

Desenvolvido em React com Vite, o frontend foi estruturado em componentes reutilizáveis e organizado em páginas e formulários que representam cada funcionalidade principal: gerenciamento de ACLs, regras de cache, regras de banda, autenticação, visualização do servidor e navegação entre as áreas da aplicação.

A logo do sistema (presente na página do navegador, página inicial e barra de navegação da interface), foi elaborada visando: simplicidade, modernidade e remetimento ao servidor Squid. As cores predominantes nas telas seguiram a mesma ideia, sendo elas o azul marinho, cinza e branco. A figura 6 demonstra a logo do sistema:

Figura 6 – Logo do Sistema



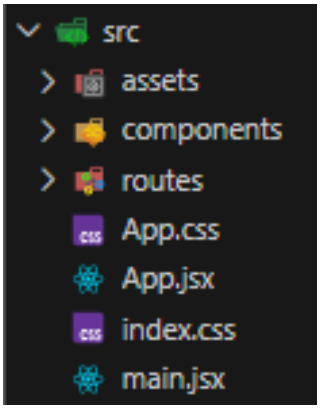
Fonte: Elaboração própria.

A estrutura geral do frontend segue um padrão modular e organizado, dividido em três diretórios principais:

- *components/* – componentes reutilizáveis, como formulários, botões, descrições e itens comuns entre páginas;
- *routes/* – arquivos responsáveis por definir as rotas e proteger páginas privadas;
- *assets/* – imagens e recursos estáticos;
- Arquivos *Page.jsx* – representam páginas completas do sistema, cada uma contendo lógica própria de interação com backend e apresentação visual.

Além disso, arquivos globais como *App.jsx, main.jsx* e index.css controlam a inicialização e o estilo geral da aplicação. Pode-se observar essa estrutura de arquivos na Figura 7.

Figura 7 – Estrutura dos Arquivos do Frontend



Fonte: Elaboração própria.

4.4.1.1 Routes

Cada funcionalidade principal do sistema foi isolada em uma página dedicada, permitindo clareza na navegação e separação de responsabilidades:

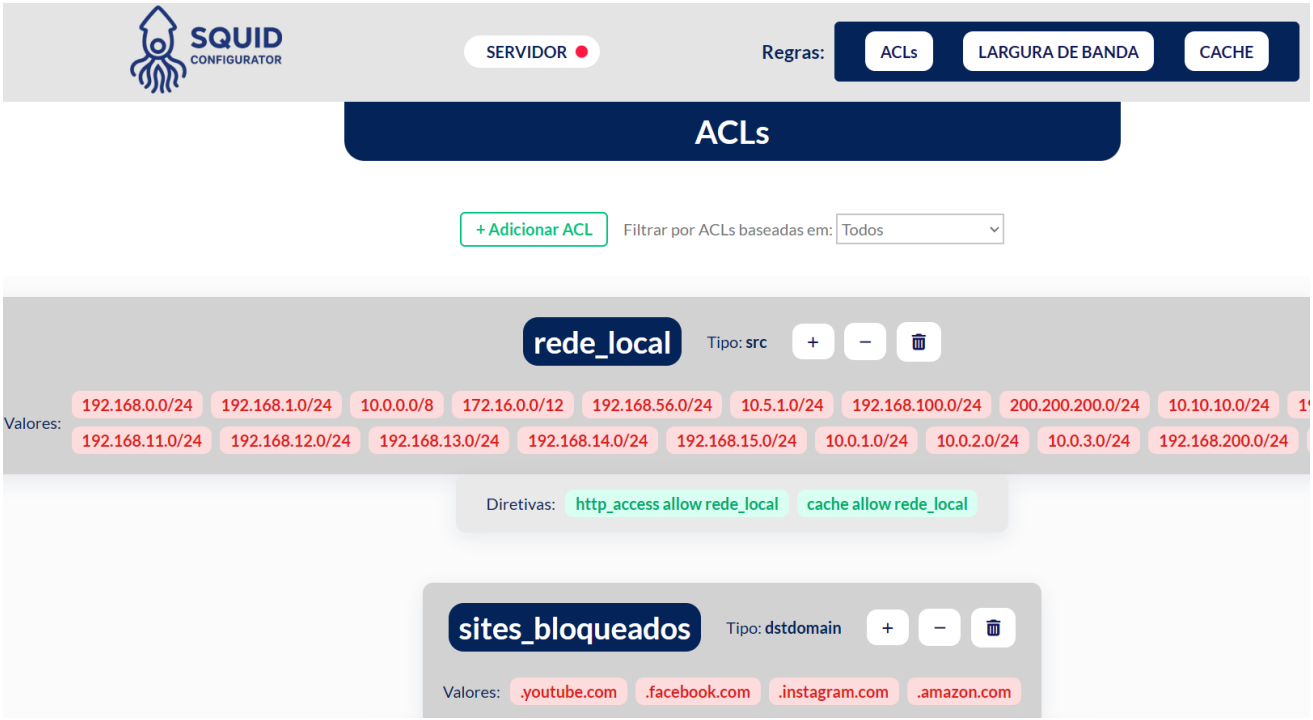
- *AclsPage.jsx* – exibe ACLs existentes, permite criação, edição e remoção de valores e diretivas;

- *CacheRulesPage.jsx* – manipula regras de cache com formulários específicos;
- *BandWidthRulesPage.jsx* – controla configuração de regras de banda;
- *ServerPage.jsx* – monitora o estado do Squid e executa ações como iniciar, parar e recarregar o serviço;
- *HomePage.jsx* – apresenta a página inicial do sistema com informações gerais;
- *LoginPage.jsx* – responsável pela autenticação do usuário.

Cada página combina elementos visuais (CSS próprio) com lógica interna para chamadas ao backend, atualizações de estado e validações.

A navegação foi implementada utilizando *React Router*: a *main.jsx* contém o *PrivateRoute.jsx*, componente responsável por proteger páginas. Quando se tenta acessar o sistema, o *PrivateRoute* garante que apenas usuários autenticados possam visualizar o conteúdo, redirecionando automaticamente para a página de login quando necessário. A Figura 8 trata-se de uma demonstração de página:

Figura 8 – Barra de Navegação entre Páginas e Página de Exemplo (ACLs)



Fonte: Elaboração própria.

Todas as páginas da interface dotam de uma barra de navegação e de um *footer*. A Figura 8 demonstra a página de ACLs e seu padrão de exibição segue o mesmo nas páginas de Largura de Banda e Cache. Em cada uma está presente um botão para adição de uma nova regra e uma lista contendo as regras daquele respectivo tipo, assim como (no caso das ACLs) seus valores e diretivas já associadas. Para cada uma das regras existem botões para: adicionar valor/diretiva, remover valor/diretiva e excluir regra, ação que possui confirmação via navegador.

4.4.1.2 Components

O diretório *components/* concentra estruturas reutilizáveis que garantem padronização visual e reduzem duplicação de código. Entre os principais componentes estão os presentes na Tabela 1:

Tabela 1 – Organização dos componentes do frontend

Categoria	Componentes	Função
Formulários específicos	***AclsForm.jsx***, ***ValuesForm.jsx***, ***CacheRulesForm.jsx***, ***BandWidthRulesForm.jsx***, ***AclDirectivesForm.jsx***	Responsáveis pela criação, edição e manipulação detalhada das regras e diretivas do Squid. Cada formulário trata de um tipo particular de configuração, guiando o usuário na construção correta das regras.
Componentes de UI	***Button.jsx***, ***Select.jsx***, ***NavBar.jsx***, ***Footer.jsx***	Componentes reutilizáveis que garantem consistência visual e comportamental no sistema. Padronizam botões, seletores, navegação e rodapé.

Categoria	Componentes	Função
Componentes auxiliares	***DirectiveOrValueChoiceForm.jsx***	Componente intermediário que auxilia o usuário na escolha entre diferentes formas de inserir regras (diretivas diretas ou valores associados). Atua como um passo lógico dentro dos formulários principais.

Fonte: Elaboração própria.

Cada componente possui também um arquivo *.css* dedicado, permitindo estilização modular sem interferir nas demais partes da aplicação.

Toda interação entre o frontend e o backend é realizada por meio da API REST exposta pelo Spring Boot. As páginas utilizam *fetch* para executar operações como: listar ACLs, adicionar regras, remover valores, atualizar diretivas, verificar o estado do Squid, enviar comandos de iniciar/parar/recarregar o serviço, etc. As respostas do backend são utilizadas para atualizar imediatamente o estado dos componentes React, garantindo que a visualização seja atualizada assim que o arquivo *squid.conf* seja modificado.

4.4.2 Backend

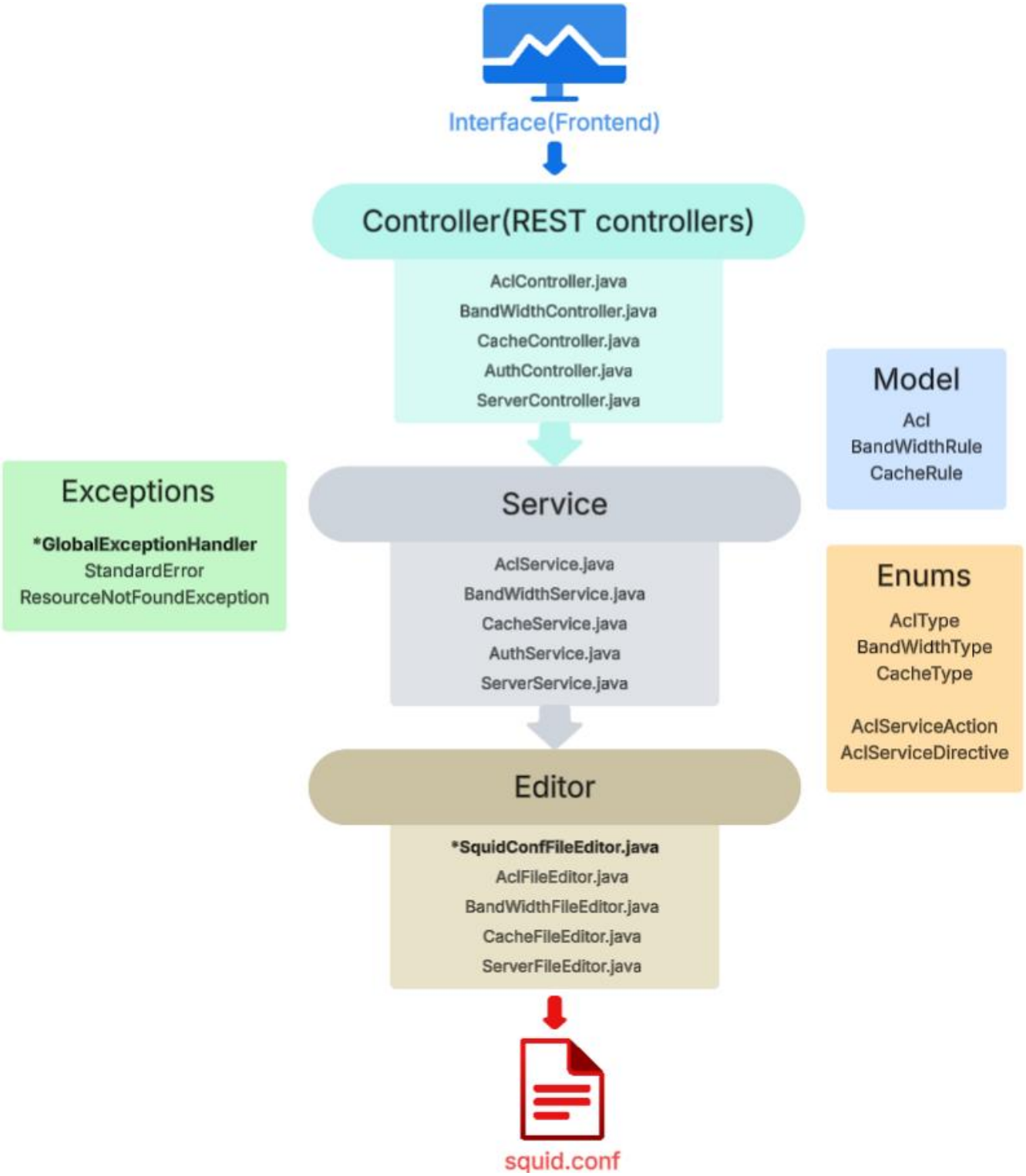
A implementação do backend deste sistema foi construída sobre o ecossistema Spring, utilizando Java como linguagem principal. A arquitetura desenvolvida organiza o código em camadas bem definidas (controle, serviço e manipulação de arquivos) permitindo que cada parte do sistema mantenha responsabilidades claras e independentes. O backend é responsável por receber requisições do frontend, processar comandos de edição no arquivo *squid.conf*, validar entradas, manipular regras específicas e garantir respostas consistentes à interface. Para isso, foram implementados controladores REST, serviços especializados e classes de edição de arquivos que traduzem as operações solicitadas pelo usuário em modificações estruturadas no arquivo de configuração do Squid. Além disso, o sistema conta com

um mecanismo centralizado de tratamento de exceções, padronizando erros e garantindo seu funcionamento correto.

4.4.2.1 Arquitetura

A arquitetura implementada neste projeto segue o modelo clássico de camadas, no qual cada parte do sistema possui responsabilidades bem definidas e independentes entre si. A camada de apresentação é composta pelos *Controllers*, responsáveis por receber as requisições HTTP, interpretar os dados enviados pelo frontend e encaminhá-los para o nível seguinte. A camada de serviço centraliza as regras de negócio da aplicação, coordenando o fluxo entre a API e as operações de manipulação do arquivo de configuração do Squid. Já a camada de acesso ao recurso, representada pelas classes *Editor*, realiza diretamente as operações de leitura, escrita e atualização no arquivo *squid.conf*, garantindo que as modificações solicitadas sejam executadas de forma consistente e segura. Essa separação clara favorece organização, testabilidade, evolução do código e redução de acoplamento entre as partes do sistema e pode ser observada na Figura 9. Em seguida dela, será possível observar essas camadas a nível de classes num diagrama UML (Figura 10).

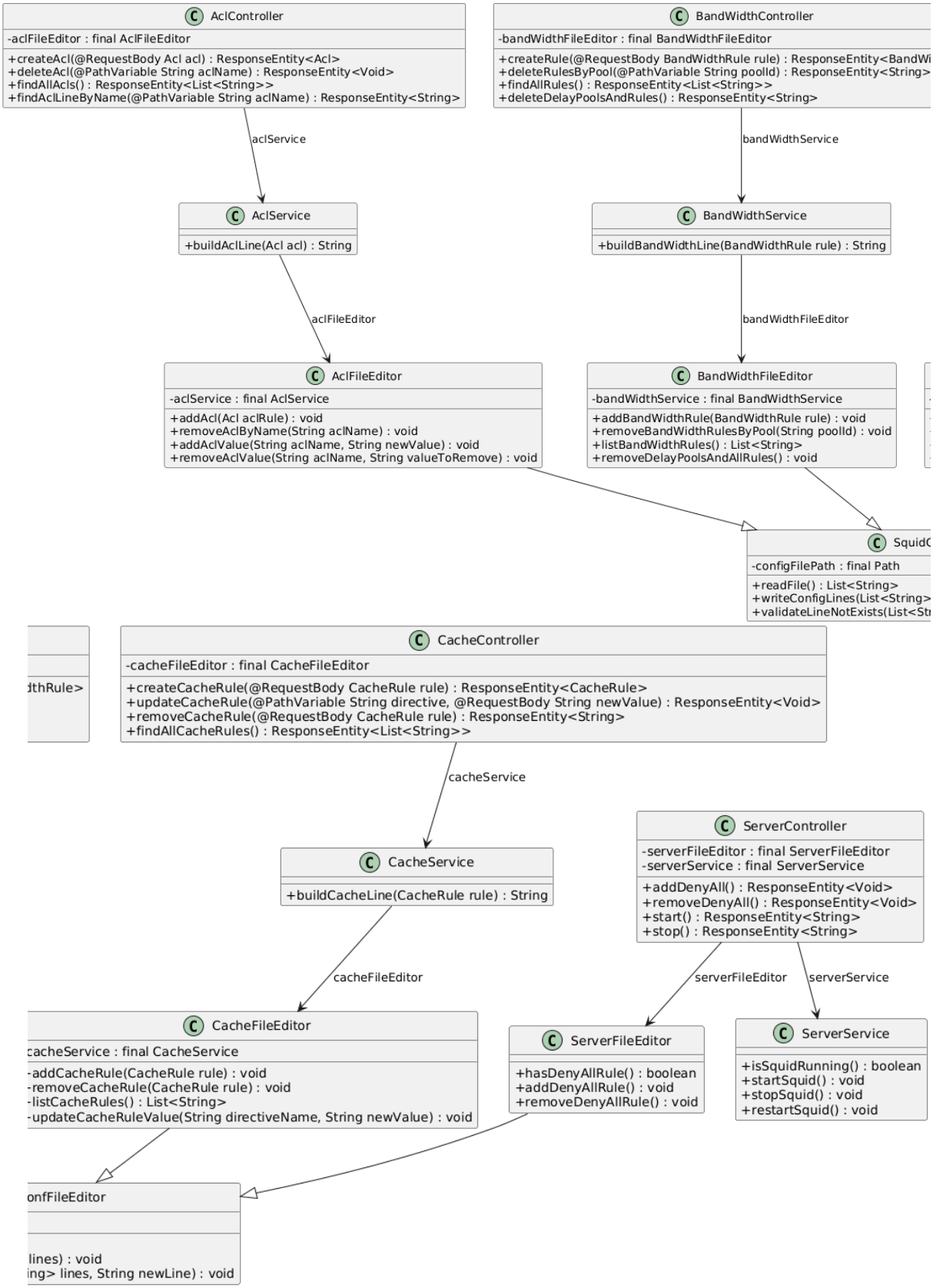
Figura 9 – Arquitetura em Camadas da Aplicação



Fonte: Elaboração própria.

4.4.2.2 Implementação e Interação de Classes

Figura 10 – Diagrama de Classes UML



Fonte: Elaboração própria.

É importante salientar que o diagrama acima está simplificado (não mostra todas as classes e métodos) por questões de espaço, mas mostra o fluxo principal das interações do usuário com o arquivo de configuração.

Antes de adentrar em mais detalhes de implementação, é pertinente que se realize algumas observações acerca das *services* e suas interações dentro do sistema:

1. As Services da forma que estão implementadas não fazem parte diretamente do fluxo das requisições (não recebem requisições nem operam no arquivo), porém servem como classes auxiliares das *Editors* na montagem das linhas de suas respectivas regras, de acordo com suas regras de negócio, contribuindo mais em separação de responsabilidades;

2. Como as regras de Cache e Largura de Banda não possuem outros componentes associados (como múltiplos valores ou diretivas das ACLs), suas Services são mais simples e possuem apenas um método de montagem da linha cada uma;

3. A *ServerService* funciona de forma diferente das outras. Por ser responsável apenas de realizar os comandos de funcionamento do Squid (iniciar, reiniciar, etc.), ela não interage com o *ServerFileEditor*, mas centraliza os métodos que realizam esses comandos diretamente no S.O.

A seguir, para explicitar um dos braços do fluxo principal, serão apresentadas as partes principais da *controller*, *service* e *editor* das ACLs nas figuras 11 e 12:

Figura 11 – AclController e AclService

```java
@CrossOrigin(origins = "http://localhost:5173")
@RestController
@RequestMapping("/acls")
public class AclController {

    private final AclFileEditor aclFileEditor;

    public AclController(AclFileEditor aclFileEditor) {
        this.aclFileEditor = aclFileEditor;
    }

    @PostMapping
    public ResponseEntity<Acl> createAcl(@RequestBody Acl acl) throws IOException {
        aclFileEditor.addAcl(acl);
        URI uri = ServletUriComponentsBuilder
                    .fromCurrentRequest()
                    .path("/{name}")
                    .buildAndExpand(acl.getName())
                    .toUri();
        return ResponseEntity.created(uri).body(acl);
    }


    @DeleteMapping("/{aclName}")
    public ResponseEntity<Void> deleteAcl(@PathVariable String aclName) throws IOException {
        aclFileEditor.removeAclByName(aclName);
        return ResponseEntity.noContent().build();
    }

    @GetMapping
    public ResponseEntity<List<String>> findAllAcls() throws IOException {
        List<String> rules = aclFileEditor.listAclRules();
        return ResponseEntity.ok(rules);
    }
```

```java
public class AclService {

    public String buildAclLine(Acl acl) {
        String name = "acl " + acl.getName().trim();
        String value = acl.getValue();
        switch (acl.getAclType()) {
        case SRC:
            return name + " src " + value;
        case DST:
            return name + " dst " + value;
        case DSTDOMAIN:
            return name + " dstdomain " + value;
        case PORT:
            return name + " port " + value;
        case TIME:
            return name + " time MTWHF " + value;
        case URL_REGEX:
            return name + " url_regex -i " + value;
        default:
            throw new IllegalArgumentException("Tipo de ACL não suportado: " + acl.getAclType());
        }
    }

    public String addAclLineValue(String line, String newValue) {
        return line + " " + newValue;
    }

    public String removeAclLineValue(String line, String removedValue) {
        StringBuilder resultado = new StringBuilder();
        String[] values = line.trim().split("\\s+");

        for (String value : values) {
            if (!value.equals(removedValue)) {
                resultado.append(value).append(" ");
            }
        }
        return resultado.toString().trim();
    }
```

Fonte: Elaboração própria.

Figura 12 – AclFileEditor

```
@Component
public class AclFileEditor extends SquidConfFileEditor{

    private final AclService aclService = new AclService();

    public AclFileEditor(Path configFilePath) {
        super(configFilePath);
    }


    public void addAcl(Acl aclRule) throws IOException {
        String newLine = aclService.buildAclLine(aclRule);
        List<String> lines = readFile();
        boolean exists = lines.stream().anyMatch(line -> line.startsWith("acl " + aclRule.getName() + " "));
        if (exists) {
            throw new IllegalArgumentException("ACL com nome '" + aclRule.getName() + "' já existe.");
        }
        int insertIndex = -1;
        for (int i = 0; i < lines.size(); i++) {
            if (lines.get(i).startsWith("acl ")) {
                    insertIndex = i;
            }
        }
        if (insertIndex == -1) {
            lines.add(0, newLine);
        } else {
        lines.add(insertIndex + 1, newLine);
        }
        writeConfigLines(lines);
    }
```

Fonte: Elaboração própria.

Será colocada a seguir uma breve descrição a respeito da atuação de cada uma das classes apresentadas nas figuras 11 e 12:

- A *AclController* é a camada responsável por expor os *endpoints* REST relacionados às ACLs (métodos *createAcl*, *deleteAcl*, etc). Ela recebe as requisições do frontend, interpreta os parâmetros enviados e coordena as operações necessárias chamando os serviços apropriados. Seu papel é atuar como intermediário entre a interface do usuário e a lógica de negócio, garantindo que as operações de criação, leitura, atualização e exclusão de ACLs sejam acessadas de forma padronizada e segura;

- A *AclService* concentra a lógica de negócio associada ao gerenciamento das ACLs. Ela valida os dados recebidos, aplica regras de consistência e decide como cada operação deve ser processada antes de interagir com o arquivo de configuração. Além disso, o serviço encapsula as regras que determinam como uma ACL deve ser estruturada e integrada ao arquivo, montando sua respectiva linha em momentos de criação e de edições de uma linha já existente (como

em adições/remoções de valores, com os métodos *addAclLineValue* e *removeAclLineValue*);

- A *AclFileEditor* é a classe responsável por acessar diretamente o arquivo *squid.conf* e aplicar as modificações necessárias de ACLs. Ela executa operações de leitura e escrita no arquivo de configuração, localiza os locais corretos de inserção (como observado em addAcl), insere ou remove regras e garante que o conteúdo final esteja formatado conforme a sintaxe do Squid. Essa classe representa a camada de manipulação de dados persistidos, sendo crucial para que as alterações solicitadas pelo sistema sejam refletidas no ambiente real de configuração do proxy.

É possível observar pela Figura 12 e pelo Diagrama UML (Figura 11) que todas as *Editors* estendem de uma principal, a *SquidConfigEditor* (Figura 13), funcionando da seguinte forma: enquanto nas *Editors* especificas estão as normas de inserção/edição no arquivo respectivas de cada regra (onde inserir a linha, se existe uma linha conflitante, se já existe uma linha necessária para aquela poder ser adicionada, entre outras), na *Editor* principal estão os métodos que de fato leem o arquivo e realizam as inserções.

Figura 13 – SquidConfFileEditor

```
8  public class SquidConfFileEditor {
9
10     private final Path configFilePath;
11
12     public SquidConfFileEditor(Path configFilePath) {
13         this.configFilePath = configFilePath;
14     }
15
16     protected List<String> readFile() throws IOException {
17         return Files.readAllLines(configFilePath);
18     }
19
20     protected void writeConfigLines(List<String> lines) throws IOException {
21         Files.write(configFilePath, lines);
22     }
23
24     protected void validateLineNotExists(List<String> lines, String newLine) {
25         if (lines.contains(newLine)) {
26             throw new IllegalArgumentException("Regra já existe no arquivo.");
```

Fonte: Elaboração própria.

4.4.2.3 Classes de Suporte

Existem classes de suporte que não fazem parte da arquitetura principal, mas são essenciais para fornecer recursos que possibilitam o funcionamento correto do sistema. A seguir será colocada em formato de tabela (Tabela 2) uma descrição das classes de suporte presentes na aplicação:

Tabela 2 – Classes de Suporte do Backend

Categoria	Classes	Função
Configuração e Injeção de Dependências	*SquidConfig.java*	Responsável por definir *beans* essenciais do sistema, como o caminho do arquivo *squid.conf* e instâncias compartilhadas dos editores. Centraliza a configuração e fornece dependências para serviços e controladores.
Tratamento de Exceções	*ResourceNotFoundException.java* *StandardError.java* *GlobalExceptionHandler.java*	Classes destinadas a representar erros específicos do domínio, encapsular as exceções em objetos e interceptar os erros do sistema transformando em respostas HTTP padronizadas ao usuário.
Autenticação	*AuthController.java* *AuthService.java*	Implementam a autenticação básica do backend. O controlador recebe as credenciais enviadas pelo frontend, enquanto o serviço valida usuário e senha.

Fonte: Elaboração própria.

5 PROCEDIMENTOS VALIDAÇÃO

Os procedimentos de validação têm como objetivo assegurar que o sistema desenvolvido funcione corretamente tanto no nível lógico quanto no ambiente operacional real. Para isso, os testes foram conduzidos em dois ambientes distintos:

1. Um ambiente interno de testes, utilizando um arquivo configurado exclusivamente para experimentação dentro do próprio projeto;

2. Um ambiente virtual completo, composto por um servidor executando o Squid e por uma máquina cliente submetida às regras impostas pelo proxy.

A separação desses ambientes se justifica pela necessidade de avaliar aspectos diferentes do sistema. Essa abordagem dupla garante que o sistema seja validado tanto em sua lógica interna quanto em sua aplicação prática, proporcionando maior confiabilidade aos resultados.

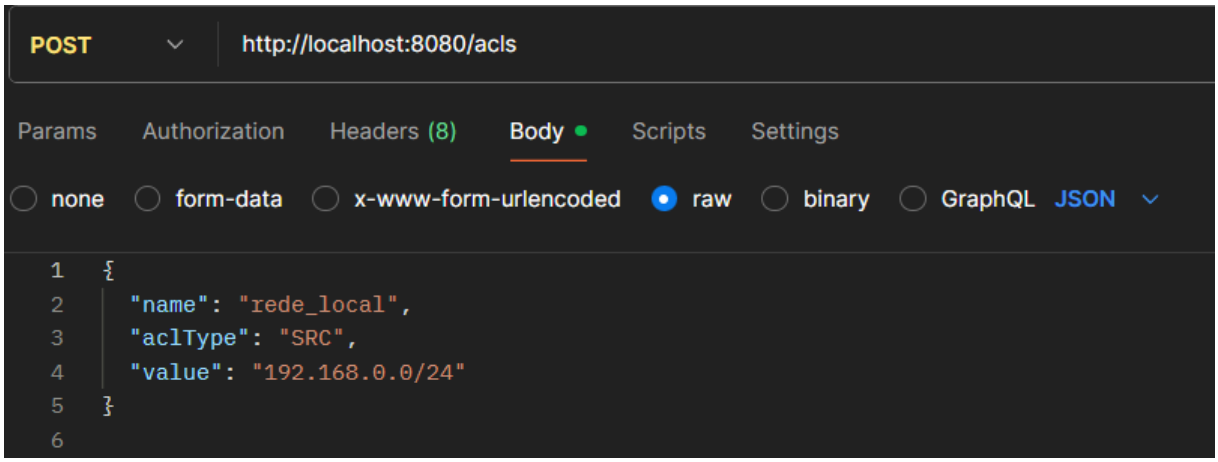
5.1 Testes no Ambiente Interno

O ambiente interno de testes foi projetado para validar o funcionamento lógico dos módulos principais do sistema (controladores, serviços e editores) sem depender diretamente do servidor Squid ou de um ambiente de rede completo. Nesse contexto, foi utilizado um arquivo de configuração dedicado exclusivamente aos testes, localizado dentro do próprio projeto (*teste_squid.conf*). Esse arquivo funcionou como uma simulação isolada do *squid.conf*, permitindo avaliar operações de leitura, escrita e atualização de forma controlada e repetível.

Os testes foram executados por meio do *Postman*, ferramenta que possibilita o envio estruturado de requisições HTTP para a API. Essa abordagem permitiu verificar se os endpoints respondiam corretamente (como na Figura 14), garantindo que cada alteração solicitada ao backend atualizasse o arquivo de teste de acordo com a lógica

implementada. Com isso, foi possível identificar rapidamente inconsistências, validar fluxos e corrigir comportamentos antes de avançar para o ambiente virtual.

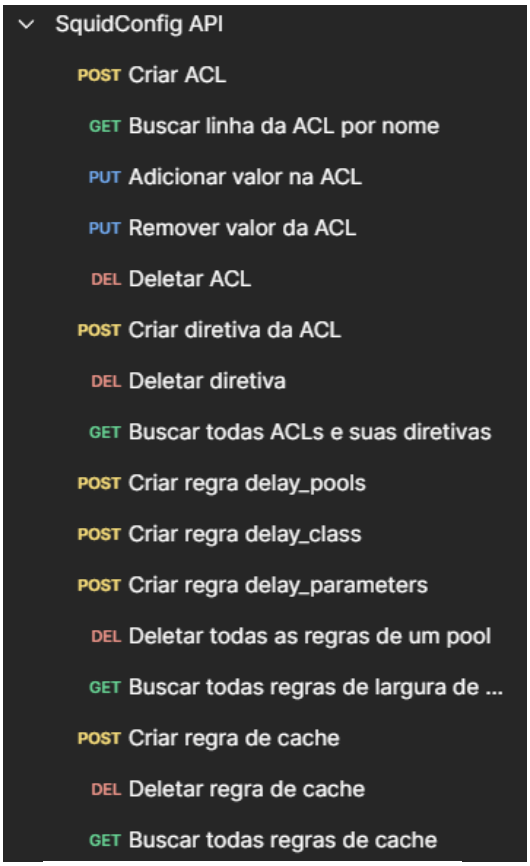
Figura 14 – Exemplo de Requisição no Postman (Criação de ACL)



Fonte: Elaboração própria.

Como resultado, foi possível confirmar o funcionamento de todas as operações presentes na *collection* criada *SquidConfigAPI* (figura 15):

Figura 15 – Collection SquidConfigAPI

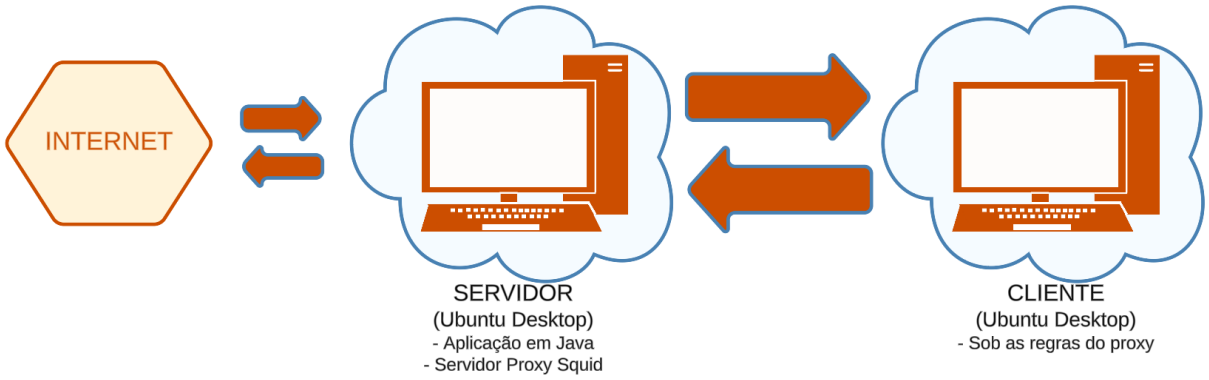


Fonte: Elaboração própria.

5.2 Testes no Ambiente Virtual

O ambiente virtual foi composto por duas máquinas virtuais Ubuntu Desktop, uma é o servidor (onde o Squid e a aplicação estão rodando) e a outra o cliente, onde foi possível observar as regras de proxy que o usuário define por meio da aplicação. Esse ambiente pode ser observado na Figura 16.

Figura 16 – Esquema de Ambiente de Testagem



Fonte: Elaboração própria.

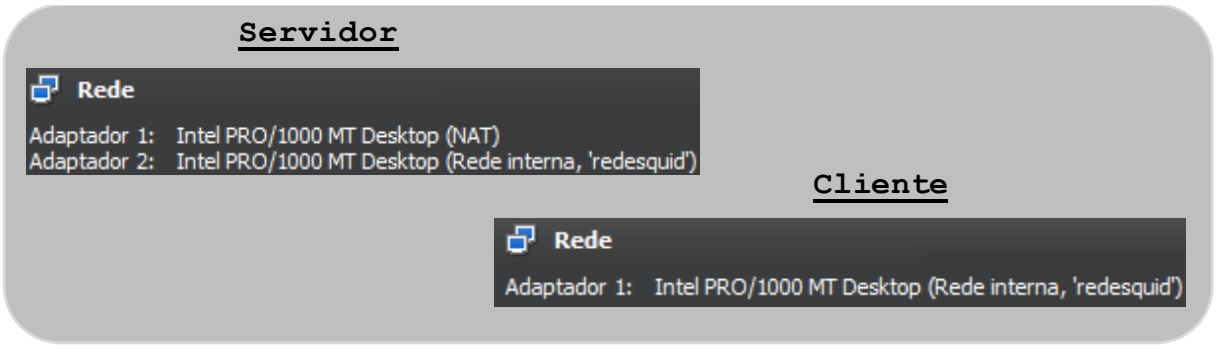
5.3 Preparação do Ambiente Virtual

Para criar e configurar as máquinas virtuais utilizadas na testagem foi usado o software Oracle VM Virtualbox.

Primeiramente, foi baixada uma ISO do S.O Ubuntu Desktop, a qual foi utilizada para atuar como o sistema operacional de ambas as máquinas. Também para ambas foram adicionados recursos computacionais suficientes para rodar o Squid e a aplicação, para assim se realizar os testes de forma estável. Em seguida, por se tratar de um ambiente que envolve um servidor e um cliente, o aspecto mais crucial: a configuração dos adaptadores de rede; precisando funcionar de forma que apenas o servidor tenha acesso direto à internet e a conexão do cliente precise passar antes por ele, os adaptadores foram estabelecidos da seguinte forma: para o servidor foram colocados dois adaptadores, o primeiro como “NAT” (Internet) e o segundo como

“Rede Interna” enquanto o cliente ficou com apenas um, como “Rede Interna” (Figura 17),.

Figura 17 – Adaptadores de Rede



Fonte: Elaboração própria.

Dessa forma, o servidor se conecta à internet e a uma rede interna (com nome “redesquid”), já o cliente se conecta apenas à rede interna e para acessar a internet precisará invariavelmente passar pelo servidor, consequentemente passando pelas regras do Squid. Feitos esses procedimentos, as máquinas estão criadas e configuradas corretamente, mas agora precisam se comunicar e terem seus comportamentos esperados estabelecidos.

Para as máquinas se encontrarem na rede foram atribuídos IPs estáticos para cada uma delas, sendo para o servidor `192.168.56.1/24` e para o cliente `192.168.56.101/24`, assim, através de um *ping* já se comunicavam. A seguir, serão colocados em passos os procedimentos realizados no servidor para o fazer atuar como roteador e no cliente para o usar como *gateway* e proxy:

1. Habilitando o encaminhamento de pacotes: no servidor, para permitir que ocorra o tráfego entre a NAT e a rede interna (onde está o cliente), é necessário acessar o arquivo de configuração do sistema com `sudo nano /etc/sysctl.conf` e habilitar o encaminhamento de pacotes apenas

descomentando a linha `net.ipv4.ip_forward=1`. Para aplicar as mudanças basta realizar `sudo sysctl -p`;

2. Permitindo atuação do NAT: é necessário adicionar a regra `sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE`, onde `enp0s3` corresponde à interface NAT (adaptador de rede 1) do servidor;

3. Persistindo a regra: para que a regra anterior persista a reinicializações, adiciona-se a configuração `sudo iptables-save | sudo tee /etc/iptables/rules.v4`;

4. Definindo *gateway*: no cliente, deve-se acessar o arquivo de configuração de rede com `sudo nano /etc/netplan/00-installer-config.yaml` e adicionar as configurações:

```
routes:
  - to: default
    via: 192.168.56.1
```

   Assim, a rota padrão de rede do cliente é definida passando pelo servidor (utilizando de seu IP). Para aplicar basta realizar `sudo netplan apply` e testar com um *ping* em algum endereço da rede;

5. Configurando o cliente para usar o proxy: para que o cliente possa acessar a rede por meio do Squid rodando no servidor, é necessário com `nano ~/.bashrc` acessar o arquivo e adicionar ao seu final a linha `export http_proxy=http://192.168.56.1:3128`;

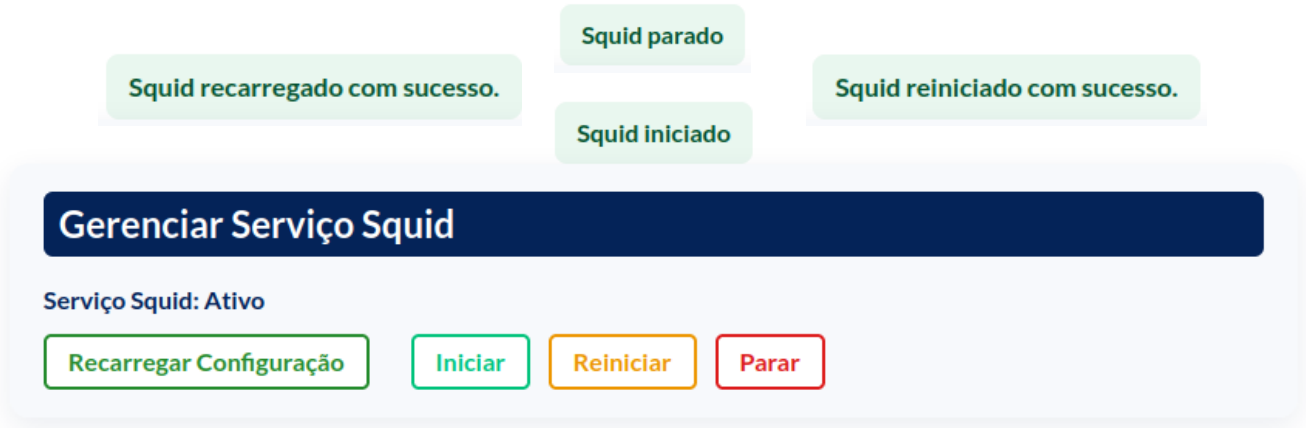
6. Configurando navegadores: alguns navegadores possuem próprias configurações de proxy, como o Mozilla Firefox. Nesses casos, é necessário que se entre nas configurações manuais de proxy do navegador e insira o endereço de proxy (nesse caso 192.168.51.1) e a porta (nesse caso 3128).

Após realizados esses procedimentos, foram realizados os passos apresentados no capítulo “Funcionamento e Configuração Básica do Squid”, para configurar o servidor proxy. Assim, o ambiente de testagem estava pronto e funcional para se experimentar as configurações e funcionalidades oferecidas pelo Squid e o desempenho da aplicação.

5.4 Resultados

Durante os testes, o sistema gerou todos os diferentes tipos de diretivas e ACLs, que foram aplicadas diretamente ao arquivo squid.conf do servidor virtual. Após cada modificação, foram executados comandos operacionais do Squid como inicialização, reinicialização e verificação de status, a fim de garantir que as alterações fossem aceitas e que o serviço continuasse funcionando corretamente. Esse processo permitiu identificar configurações inválidas, verificar mensagens de erro fornecidas pelo Squid e confirmar que as regras geradas pela aplicação estavam sintaticamente corretas. Os resultados da realização dos comandos são atestados pela própria aplicação (Figura 18):

Figura 18 – Feedback do Sistema aos Comandos de Gerenciamento do Squid



Fonte: Elaboração própria.

O comportamento da máquina cliente serviu como principal indicador da eficácia das configurações aplicadas. Testes foram realizados acessando páginas permitidas, bloqueadas ou controladas de acordo com as ACLs definidas. Quando as

regras eram aceitas pelo Squid e aplicadas corretamente, o cliente respondia exatamente conforme o esperado, validando o funcionamento real das diretivas criadas pelo sistema. A seguir será apresentado um exemplo do funcionamento de uma ACL gerenciada a partir do sistema (Figura 19):

Figura 19 – Demonstração de Uso de ACL no Sistema

Fonte: Elaboração própria.

A Figura 19 pode ser dividida em duas partes: uma demonstra a ACL “sites_bloqueados” gerenciada pelo sistema e a outra a visão do cliente sob as regras do Squid. A ACL é do tipo *dstdomain* (domínio de destino) e possui um valor referente ao site do Facebook. A diretiva adicionada (*http_access deny*) indica que o acesso via

http ao domínio do Facebook está bloqueado. Ao tentarmos acessar o Facebook pelo cliente, atestamos o funcionamento da regra ao receber a recusa do acesso pelo proxy.

De forma geral, os resultados demonstraram que a aplicação foi capaz de gerar configurações válidas, aplicar as mudanças no servidor e produzir impacto imediato no tráfego do cliente. A interação entre a ferramenta e o Squid mostrou-se consistente, indicando que o sistema atende aos requisitos propostos para a automação das configurações no ambiente de rede.

6 CONCLUSÃO

A conclusão reúne uma síntese dos resultados e organiza observações relevantes sobre dificuldades encontradas, limitações do sistema e propostas de melhorias para evolução do projeto.

As maiores dificuldades não tiveram tanta ligação à lógica de implementação em si, mas à preparação e integração do ambiente. A configuração do Squid e do ambiente virtual, com ajustes de caminhos, permissões e versões do Squid exigiram tempo e testes repetidos para que o proxy aceitasse as modificações geradas pela aplicação. Também não foi possível testar a eficácia das regras de largura de banda pois a versão do Squid utilizada não oferecia esse suporte. Além disso, foi realizada uma grande mudança de escopo no projeto onde ele passou a dotar de uma interface, decisão que teve como principal embasamento a necessidade de uma interface intuitiva para uma configuração mais visual e menos exigente em termos de conhecimento técnico.

No quesito implementação, as principais dificuldades foram duas:

- Integração frontend e backend: divergências no formato dos dados e contratos causaram retrabalhos durante os testes, causados predominantemente pelo uso exagerado de *enums* (que poderiam ser substituídas por interfaces);

- Transições entre formulários no frontend: fluxo de estados e validações em vários formulários trouxe complexidade UX e bugs de sincronização de estado.

A versão final do sistema também apresenta algumas limitações, sendo elas:

- Associação entre ACLs e regras de banda não implementada: ainda não é possível associar explicitamente ACLs às regras de largura de banda, impedindo o funcionamento pleno desse recurso;

- Validação de valores das regras insuficiente: o sistema não valida criteriosamente os parâmetros numéricos/textuais das regras (ex.: limites de banda, apenas endereços IP para ACLs SRC), o que exige conhecimento prévio do Squid por parte do operador;

- Dependência de conhecimento humano: sem validação e sem testes automáticos da eficácia, o sistema ainda depende de um operador com experiência em Squid para garantir configurações corretas.

Tendo-se conhecimento do que o sistema apresenta de funcionalidades e recursos, assim como de limitações, pode-se elencar como melhorias futuras:

- Associação de ACLs a regras de largura de banda: permitir vincular explicitamente ACLs a políticas de banda, com interface para selecionar ACLs existentes e suas respectivas políticas;

- Controle de usuários da rede: adicionar funcionalidades de gerenciamento de usuários (cadastro, autenticação avançada, grupos) e políticas por usuário/grupo;

- Validação e sanitização de valores: implementar validações completas no backend e feedback no frontend para prevenir regras inválidas. Essas validações de backend tem espaço para serem implementadas nas *services* de cada tipo de regra, mantendo seu proposito original intacto e sem afetar o fluxo do sistema.

Como considerações finais, o projeto alcançou seu objetivo principal de automatizar a criação e aplicação de configurações do Squid em um fluxo organizado, assim como tornar o complexo processo de configuração mais intuitivo. As limitações encontradas são em grande parte relacionadas ao ambiente e à compatibilidade do proxy, e não à arquitetura do sistema. As recomendações e melhorias acima apontam um caminho claro para transformar a aplicação em uma ferramenta ainda mais consistente e pronta para uso em produção.

O código fonte, assim como todo histórico de versionamento dele está disponível em: https://github.com/renatofsimoes/squid-configurator.

REFERÊNCIAS

EDELMAN, Jason; LOWE, Scott; OSWALT, Matt. ***Network Programmability and Automation***. Sebastopol: O'Reilly Media, 2018. 581 p.

CURSO de VirtualBox - Completo - Configurações de Redes, NAT, Bridge, Interna e dicas extras!. [S.l.]: 4Two, 2017. (37 min.), P&B. Disponível em: https://www.youtube.com/watch?v=HncE_R7S_d4&t=898s. Acesso em: 09 set. 2024.

SQUID Proxy Instalação e Configuração (Linux). [S.l.]: Simplificando Redes, 2021. (26 min.), P&B. Disponível em: https://www.youtube.com/watch?v=jc1dljp9T1s&t=147s. Acesso em: 21 set. 2024.

CAMERON, Jamie; COOPER, Joe. ***Squid Proxy Server***. 2023. Disponível em: https://webmin.com/docs/modules/squid-proxy-server/. Acesso em: 09 set. 2024.

SAINI, Kulbir. ***Squid Proxy Server 3.1****: improve the performance of your network using the caching and access control capabilities of squid*. Birmingham: Packt Publishing, 2011. 308 p.

SQUID CACHE. ***Squid-cache***. Disponível em: https://www.squid-cache.org/. Acesso em: 04 nov. 2024.

UBUNTU SERVER. ***How to install a Squid server***. Disponível em: https://documentation.ubuntu.com/server/how-to/web-services/install-a-squid-server/?_ga=2.209296219.871200643.1730759923-1010122398.1725024291. Acesso em: 04 nov. 2024.

WESSELS, Duane. ***Squid: The Definitive Guide***. Sebastopol: O'Reilly Media, 2004. 770 p.