UNIVERSIDADE FEDERAL DO PAMPA

Rodrigo Cargnelutti

Uma API Gateway para Controle de Acesso baseado em Arquitetura para Microsserviços

Alegrete
2024

Rodrigo Cargnelutti

Uma API Gateway para Controle de Acesso baseado em Arquitetura para Microsserviços

Trabalho de dissertação apresentado ao Programa de Pós-Graduação em Engenharia de Software como requisito parcial para a obtenção do título de Mestre em Engenharia de Software.

Orientador: Prof. Dr. Maicon Bernardino da Silveira

Alegrete
2024

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

C969a Cargnelutti, Rodrigo
Uma API Gateway para Controle de Acesso baseado em Arquitetura para Microsserviços / Rodrigo Cargnelutti.
98 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA DE SOFTWARE, 2024.
"Orientação: Maicon Bernardino da Silveira".

1. API Gateway. 2. Controle de Acesso. 3. Arquitetura de Microsserviços. 4. Revisão Sistemática da Literatura. 5. Avaliação de Desempenho. I. Título.

RODRIGO CARGNELUTTI

UMA API GATEWAY PARA CONTROLE DE ACESSO BASEADO EM ARQUITETURA PARA MICROSSERVIÇOS

Dissertação apresentada ao Programa de Pós Graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia de Software.

Dissertação defendida e aprovada em: 01/11/2024

Banca examinadora:

_____

Prof. Dr. Maicon Bernardino da Silveira

Orientador

(Unipampa)

_____

Prof. Dr. Fábio Paulo Basso

(Unipampa)

_____

Prof. Dr. Silvio Ereno Quincozes

(Unipampa)

Prof. Dr. Eduardo Kessler Piveta

(UFSM)

Assinado eletronicamente por **SILVIO ERENO QUINCOZES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 01/11/2024, às 16:16, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **FABIO PAULO BASSO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 01/11/2024, às 16:17, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MAICON BERNARDINO DA SILVEIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 01/11/2024, às 16:54, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Eduardo Kessler Piveta**, **Usuário Externo**, em 01/11/2024, às 18:24, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1582470** e o código CRC **66939754**.

Dedico esta Dissertação às minhas filhas gêmeas, Laura e Sofia, pela compreensão dos momentos de ausência durante a elaboração deste trabalho, que também é uma conquista de vocês.

AGRADECIMENTOS

À minha esposa, Daniela Cherobini Cargnelutti, por seu companheirismo e pelo apoio ao longo desta jornada.

Às minhas filhas, Laura e Sofia, pela alegria e carinho que sempre trouxeram à minha vida.

Ao meu orientador, Prof. Dr. Maicon Bernardino da Silveira, agradeço pela confiança e pelas oportunidades de aprendizado ao longo desta jornada.

Aos colegas do CPD/UFSM que se dispuseram a me prestar o auxílio necessário, meu sincero agradecimento.

Aos professores integrantes da banca examinadora, Eduardo, Sílvio e Fábio, agradeço pelas valiosas contribuições e sugestões.

RESUMO

Contexto. Atualmente, a tendência na migração de sistemas se concentra na modernização de sistemas legados para arquiteturas de microsserviços. Essa abordagem, juntamente com o controle de acesso aos sistemas tem despertado crescente interesse do Centro de Processamento de Dados (CPD) da Universidade Federal de Santa Maria (UFSM). Uma *Application Programming Interface* (*API*) *Gateway* busca modernizar a arquitetura dos sistemas, melhorar a gestão dos acessos e ajuda a mitigar riscos de segurança cibernética, promovendo uma comunicação mais segura e eficiente entre os sistemas institucionais.
Objetivo. Desenvolver uma *API Gateway* com base no *framework Spring Boot* e no *Java Development Kit* (JDK) 21. Fornecer um controle de acesso centralizado para os *webservices* do Sistema de Informações para o Ensino (SIE), permitindo a trocar informações de forma mais segura através da validação de autenticação por meio de um *token*.
Método. Foi realizada uma investigação por meio de uma Revisão Sistemática da Literatura (RSL) com o intuito de identificar soluções, ferramentas e tecnologias relacionadas à autenticação para sistemas de arquitetura de microsserviços. Foi também utilizado o método *Design Science Research* (DSR) para orientar o desenvolvimento da solução proposta. As etapas incluíram concepção, implementação e avaliação da *API Gateway*, com testes de desempenho comparativos.
Resultado. Com base na experiência adquirida, identificou-se que o *framework Spring Boot* e a biblioteca *Spring Cloud Gateway* são consideradas tecnologias adequadas para desenvolver uma *API Gateway* que atenda às demandas dos *Webservices* do SIE. A avaliação de desempenho mostrou que tanto a *API Gateway* desenvolvida quanto o *Kong API Gateway* apresentaram bom desempenho nos cenários e cargas de trabalho testados.
Conclusão. Foi desenvolvida uma *API Gateway* para aprimorar o controle de acesso aos *Webservices* do SIE. A avaliação de desempenho comparou essa solução com o *Kong API Gateway*, revelando que ambas são viáveis, cada uma com suas vantagens específicas em relação ao desempenho e consumo de recursos.

Palavras-Chave: arquiteturas de microsserviços; *API Gateway*; controle de acesso; RSL.

ABSTRACT

Background. Currently, the trend in system migration focuses on modernizing legacy systems to microservices architectures. This approach, along with system access control, has garnered increasing interest from the Data Processing Center (CPD) at the Federal University of Santa Maria (UFSM). An Application Programming Interface (API) Gateway aims to modernize the systems' architecture, improve access management, and help mitigate cybersecurity risks, promoting more secure and efficient communication between institutional systems.

Aims. To develop a modern API Gateway based on the Spring Boot framework and Java Development Kit (JDK) 21. Provide centralized access control for the web services of the Education Information System (SIE), allowing for more secure information exchange through authentication validation via a token.

Method. An investigation was conducted through a Systematic Literature Review (SLR) with the aim of identifying solutions, tools, and technologies related to authentication for microservices architecture systems.

Results. Based on the experience gained, it was identified that the Spring Boot framework and the Spring Cloud Gateway library are considered suitable technologies for developing an API Gateway that meets the demands of SIE web services.

Conclusions. An API Gateway was developed to enhance access control for the SIE web services. The performance evaluation compared this solution with the Kong API Gateway, revealing that both are viable, each with its specific advantages regarding performance and resource consumption.

Keywords: microservices architectures; API Gateway; access control; SLR.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIAÇÕES

API *Application Programming Interface*
BPMN *Business Process Model and Notation*
CISC Centro Integrado de Segurança do Governo Digital
CMS *Content Management System*
CPD Centro de Processamento de Dados
CPU *Central Processing Unit*
DSL *Domain Specific Languages*
DSR *Design Science Research*
ERES Escola Regional de Engenharia de *Software*
ES Engenharia de *Software*
HTTP *Hypertext Transfer Protocol*
IP *Internet Protocol*
JDK *Java Development Kit*
JWT *JSON Web Token*
LGPD Lei Geral de Proteção de Dados
OAuth 2.0 *Open Authorization*
RBAC *Role-Based Access Control*
RSL Revisão Sistemática da Literatura
SGBD Sistema Gerenciador de Banco de Dados
SGCA Sistema Gerenciador de Controle de Acesso
SIE Sistema de Informações para o Ensino
SSO *Single Sign-On*
TI Tecnologia da Informação
UFSM Universidade Federal de Santa Maria

SUMÁRIO

1 INTRODUÇÃO

Atualmente, o Centro de Processamento de Dados (CPD) da Universidade Federal de Santa Maria (UFSM) desenvolve e mantém os sistemas institucionais: SIE, Sítio *Web* e o aplicativo *mobile* UFSM Digital. Estes sistemas foram desenvolvidos com diferentes linguagens de programação, arquiteturas e plataformas, e não se comunicam entre si, a não ser, por meio do banco de dados e por meio dos *webservices*.

Durante anos, predominou a programação com *Delphi 7* para o desenvolvimento do SIE. A partir do ano de 2003 iniciou o desenvolvimento do SIE *Web*, com a linguagem Java 11 e atualmente é utilizado o *framework Spring Boot*. Agilar (2016) descreve em seu trabalho que Instituições Federais de Ensino Superior (IFES) estão passando pelas mesmas questões de modernização de seus sistemas legados. Um exemplo deste contexto é a UFSM, que está experimentando uma abordagem orientada a serviços para atualizar o SIE. O aplicativo UFSM Digital foi inicialmente desenvolvido com o *framework NativeScript*. No entanto, em março de 2024, o CPD lançou uma nova versão do aplicativo, desta vez utilizando a plataforma *Flutter*. O sítio *Web* é desenvolvido com o *Content Management System* (CMS) de código aberto *Wordpress*.

O SIE é um sistema integrado que abrange várias áreas e contempla diversos módulos. É amplamente utilizado em toda a UFSM e essencial para operações fundamentais da Instituição. Atualmente, o SIE contempla mais de 900 aplicações, 1500 relatórios e também possui 52 portais *Web*. O SIE está em constante evolução e novos sistemas são desenvolvidos em forma de portais *Web* (UFSM/CPD, 2024).

O aplicativo para dispositivos móveis UFSM Digital foi desenvolvido com o objetivo de disponibilizar vários serviços e facilitar o dia a dia dos estudantes, professores e dos técnicos administrativos em educação da Instituição. O sítio *Web* oferece uma vasta gama de informações sobre a UFSM, suas formas de ingresso, ensino, pesquisa e extensão, os *campi*, pró-reitorias, unidades de ensino, estrutura, editais, documentos, processos, além de diversas outras matérias e notícias. O CPD enfrenta o desafio de garantir uma abordagem que possibilite a comunicação (troca de informações) entre os *webservices* de forma segura, essencial para a modernização dos sistemas.

Os sistemas *Web* da UFSM foram submetidos a um teste de segurança *white-box*, realizado pelo Centro Integrado de Segurança do Governo Digital (CISC)[1]. O CISC oferta uma série de serviços para as instituições federais. O teste de intrusão e análise de vulnerabilidades foi aplicado para avaliar a segurança das aplicações *Web* em produção, como sites, portais do SIE e sistemas dentro do domínio da UFSM. O objetivo principal do teste de intrusão foi identificar falhas de segurança que pudessem resultar em um comprometimento da aplicação ou vazamento de dados. O relatório fornecido pela equipe do CISC contempla os resultados obtidos durante a análise, juntamente com sugestões de melhorias e recomendações.

[1] CISC: <https://www.gov.br/cisc/pt-br>

Os resultados apresentados no relatório dos testes de segurança evidenciaram a necessidade de aprimorar os aspectos de controle de acesso aos portais de *webservices* do SIE. Diante disso, optou-se neste estudo por desenvolver uma *API Gateway* para ajudar a mitigar vulnerabilidades e reforçar o controle de acesso dos portais de *webservices* do SIE.

Uma *API Gateway* serve como ponto de entrada unificado do sistema e também é um excelente método para aplicar uma camada de segurança (RAJ; VANGA; CHAUDHARY, 2023). O desenvolvimento de um aplicativo monolítico clássico comparado com o objetivo de um aplicativo na arquitetura de microsserviço envolve a reutilização de componentes de software e serviços existentes, os quais são integrados por meio de uma *API Gateway* (SNGER; ABECK, 2022). A popularidade da arquitetura de microsserviços no desenvolvimento de software tem crescido, principalmente por sua flexibilidade, abordagem granular e serviços fracamente acoplados (XU; JIN; KIM, 2019) e (LUZ et al., 2018).

Considerando o contexto, este estudo optou por seguir com a estratégia de desenvolver uma *API Gateway* para centralizar todas as solicitações externas direcionadas aos portais de *webservices* do SIE. Assim, disponibilizar uma estrutura de software que ofereça funcionalidades de controle de acesso aos portais de *webservices* do SIE, reduzindo os riscos de segurança cibernética relacionados ao acesso não autorizado. Espera-se que os benefícios incluam aprimoramento no controle de acesso, relacionados a autenticação e autorização centralizada, proteção contra acesso não autorizado e prevenção de uso indevido de dados.

Os portais *Webservice* e *Mobile*, responsáveis por receber requisições de sistemas externos ao SIE, como do Sitio *Web* e do aplicativo *Mobile* UFSM Digital, processá-las e fornecer as informações solicitadas, enfrentam um problema significativo de falta de validação e registro das requisições. Atualmente, podemos dizer que esses portais estão operando no escuro/as cegas, pois não há um método de validação para identificar quem irá utilizar o recurso. Além disso, não há um registro adequado de quem está realizando as requisições, resultando na disponibilização de informações sem identificação clara da origem que requisitou. Os *logs* registram um volume enorme de requisições aos portais, mas não identificam quem está fazendo essas requisições.

A justificativa para modernizar tecnologicamente os portais de *webservices* do SIE é a necessidade imediata de melhorar a gestão dos acessos. Portanto, é essencial apresentar uma proposta de modernização para as requisições aos portais de *webservices* do SIE, considerando o cenário atual.

Este trabalho tem como principal objetivo centralizar as requisições externas aos *webservices* do SIE, por meio de uma *API Gateway*, oferecendo uma solução customizada e alinhada às demandas da arquitetura dos sistemas da UFSM. Dessa forma, fornecer um controle de acesso centralizado para gerenciar, monitorar as solicitações e reduzir os riscos

de acesso não autorizado. Em síntese, a proposta é desenvolver uma *API Gateway* que possibilite os portais de *webservices* do SIE trocar informações com sistemas externos de forma mais segura por meio da validação do *token* contido no cabeçalho da requisição. Para atingir esse objetivo, foram definidos os seguintes objetivos específicos:

- Produzir uma Revisão Sistemática da Literatura (RSL) para identificar ferramentas e tecnologias sobre autenticação para sistemas de arquitetura de microsserviços;
- Propor o desenvolvimento de uma *API Gateway* com as funcionalidades de verificação do *token* de acesso, roteamento e registro das requisições;
- Fornecer uma funcionalidade de *logs* para monitoramento e rastreabilidade;
- Avaliar o impacto da *API Gateway* no tempo de acesso e desempenho das aplicações que utilizam os *webservices*.

A principal contribuição deste trabalho consiste na proposta de uma *API Gateway* para os *webservices* do SIE. Esta abordagem faz parte de um processo de modernização da arquitetura de software, abrangendo controle de acesso, roteamento, monitoramento e permitindo que os sistemas sejam independentes da linguagem de programação utilizada. Elencamos as seguintes contribuições:

- Condução de uma RSL sobre soluções de autenticação para microsserviços e artigos publicados na área;
- Um software para controle de acesso baseado em arquitetura modernizada para microsserviços;
- Reduzir os riscos de acesso a informações não autorizadas, fundamental para a segurança e conformidade com a Lei Geral de Proteção de Dados (LGPD).

Este documento está organizado da seguinte forma: O `Capítulo 2` aborda a metodologia empregada para conduz esta pesquisa. O `Capítulo 3` introduz o tema, explorando conceitos fundamentais, definições e tecnologias. O `Capítulo 4` apresenta a RSL conduzida para identificar ferramentas e tecnologias relacionadas à autenticação para sistemas de arquitetura de microsserviços. O `Capítulo 5` discute sobre a proposta e a implementação de uma *API Gateway*. O `Capítulo 6` descreve as avaliações comparativas de desempenho em cenários práticos. O `Capítulo 7` apresenta as conclusões obtidas e sugere oportunidades para trabalhos futuros.

2 METODOLOGIA

A construção do conhecimento deve ser fundamentada em um rigoroso método científico. A metodologia empregada para definir, conduzir e avaliar esta pesquisa foi baseada no método científico de (MARCONI; LAKATOS, 2017). Segundo os autores, é essencial compreender como estruturar a produção do conhecimento em etapas organizadas e sequenciais, semelhantes a um processo.

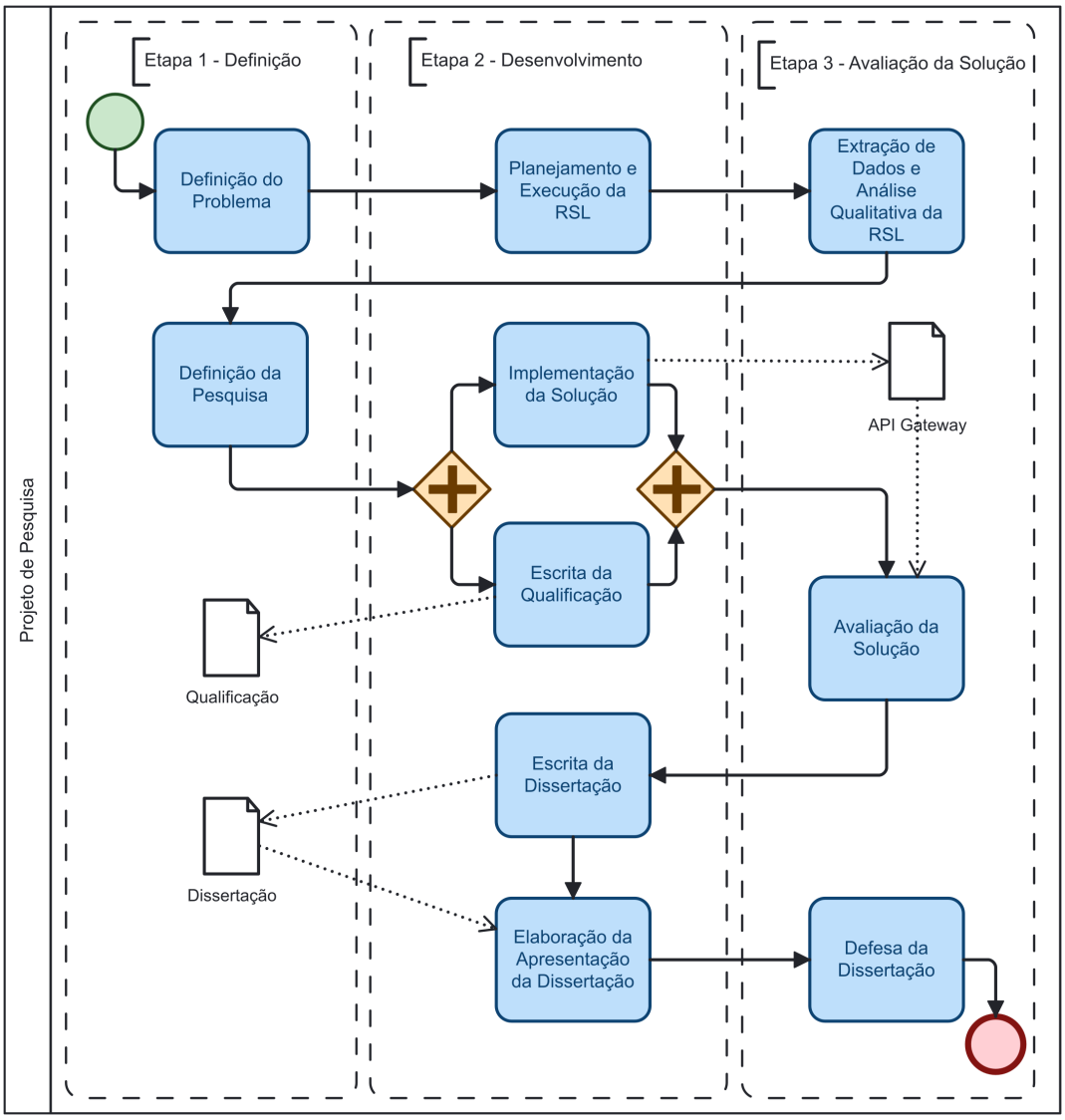
Para consolidar os conhecimentos sobre a pesquisa e desenvolver o sistema proposto foram adotadas as definições de DSR (RUNESON; ENGSTRÖM; STOREY, 2020). O processo de DSR geralmente inclui seis etapas, conforme definidas por Peffers et al. (2007): (**DSR 1**) definição do problema; (**DSR 2**) definição de objetivos para uma solução; (**DSR 3**) concepção e desenvolvimento de artefatos; (**DSR 4**) demonstração usando o artefato para resolver o problema; (**DSR 5**) avaliação da solução; e (**DSR 6**) comunicação e sua utilidade para outras pesquisas e profissionais. As pesquisas nem sempre precisam começar na primeira etapa, mas devem passar por todas as etapas de alguma maneira. O resultado de um projeto de pesquisa que segue DSR é sempre um artefato que pode ser um produto, um processo, uma tecnologia, uma ferramenta, uma metodologia, uma técnica, um procedimento, uma combinação de qualquer um desses elementos, ou qualquer outro meio para atingir um determinado propósito (VENABLE; BASKERVILLE, 2012).

A Figura 1 apresenta as atividades realizadas modeladas em *Business Process Model and Notation* (BPMN). A pesquisa iniciou pela atividade de `definição do problema`, correspondente à etapa DSR 1 da abordagem DSR. As atividades de `planejamento`, `execução`, `extração de dados`, `análise quantitativa` e a `definição da pesquisa` da RSL estão associadas à etapa DSR 2. As atividades paralelas de `implementação da solução` e de `escrita da qualificação`, `escrita da dissertação` e `elaboração da apresentação da dissertação` estão inseridas na etapa DSR 3. A atividade de `avaliação da solução` abrange as etapas DSR 4 e 5. A pesquisa é concluída com a atividade de `defesa da dissertação` correspondente à etapa DSR 6.

A experiência adquirida na disciplina de Engenharia de *Software* (ES) Experimental foi fundamental para organizar as atividades necessárias para realizar a RSL. Esta revisão foi submetida, aprovado na Escola Regional de Engenharia de *Software* (ERES) e premiada como o melhor artigo, uma distinção concedida pelo Comitê Organizador da VII ERES em conjunto com a Coordenação do Fórum de Pós-Graduação.

Seguindo a definição de Prodanov e Freitas (2013), foram estabelecidos a estrutura e o escopo de abrangência da pesquisa. Quanto a `Natureza`, trata-se de uma pesquisa `Aplicada`, voltada para geração de conhecimento com aplicação prática para a solução de problemas específicos, quanto ao `Objetivo` a pesquisa é classificada como `Exploratória`, pois visa adquirir informações para melhor compreensão de um determinado assunto e possível transferência de conhecimento. Em termos de `Procedimentos` metodológicos, está pesquisa é classificada como `Bibliográfica` devido a RSL que foi conduzida e de

Figura 1 – Desenho da Pesquisa



Fonte: Autor

`Estudo de Caso` em função da aplicação prática do conhecimento no contexto do CPD.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos teóricos abordados na realização da pesquisa deste trabalho.

3.1 Arquitetura de *Software*

A arquitetura de *software* está em constante evolução para acompanhar as demandas de desenvolvimento de sistemas, ela desempenha um papel crucial na qualidade, manutenção e escalabilidade do sistema. Além disso, é um pilar fundamental, pois molda a forma como o *software* é desenvolvido, mantido e evoluído ao longo do tempo (SORDI; MARINHO; NAGY, 2006). A arquitetura de *software* é como o esqueleto de um sistema. Ela descreve os componentes e suas funções, como o código é organizado, como os dados são armazenados e como as partes do sistema se comunicam. É o plano de mais alto nível para construir qualquer sistema (KRAFZIG; BANKE; SLAMA, 2004).

É fundamental investir em uma arquitetura de *software* sólida e bem projetada para garantir a eficiência, segurança e escalabilidade dos sistemas, pois, uma arquitetura inadequada acarreta diversos problemas tecnológicos, os quais têm impacto direto na gestão das organizações (SORDI; MARINHO; NAGY, 2006).

A modularidade, escalabilidade, flexibilidade e segurança são características essenciais da arquitetura de *software*. A arquitetura monolítica e a baseada em *microsserviços* são estilos arquiteturais que orientam a construção de um sistema.

Arquitetura Monolítica: é uma estrutura única que concentra todos os componentes, promove simplicidade, mas limita a escalabilidade. A arquitetura monolítica é um estilo de desenvolvimento de sistema em que a aplicação é construída como uma única base de código-fonte e implantada como um único pacote. Em uma aplicação monolítica, todas as funcionalidades, desde a *interface* do usuário até a lógica de negócios e a persistência de dados, são agrupadas em um único sistema (AWATI; WIGMORE, 2023);

Arquitetura baseada em Microsserviços: microsserviços é um estilo arquitetônico relativamente novo nas estratégias de desenvolvimento de software (LUZ et al., 2018). Esta arquitetura é uma abordagem emergente na Engenharia de *Software* que possibilita a construção de sistemas altamente escaláveis e flexíveis (YARYGINA; BAGGE, 2018). Na arquitetura de microsserviços um aplicativo é desenvolvido por um conjunto de pequenos serviços independentes, cada um com a sua responsabilidade e se comunicando por meio de mecanismos leves, geralmente usando *API*s baseadas em *Hypertext Transfer Protocol* (HTTP) (XIONG; LI, 2022).

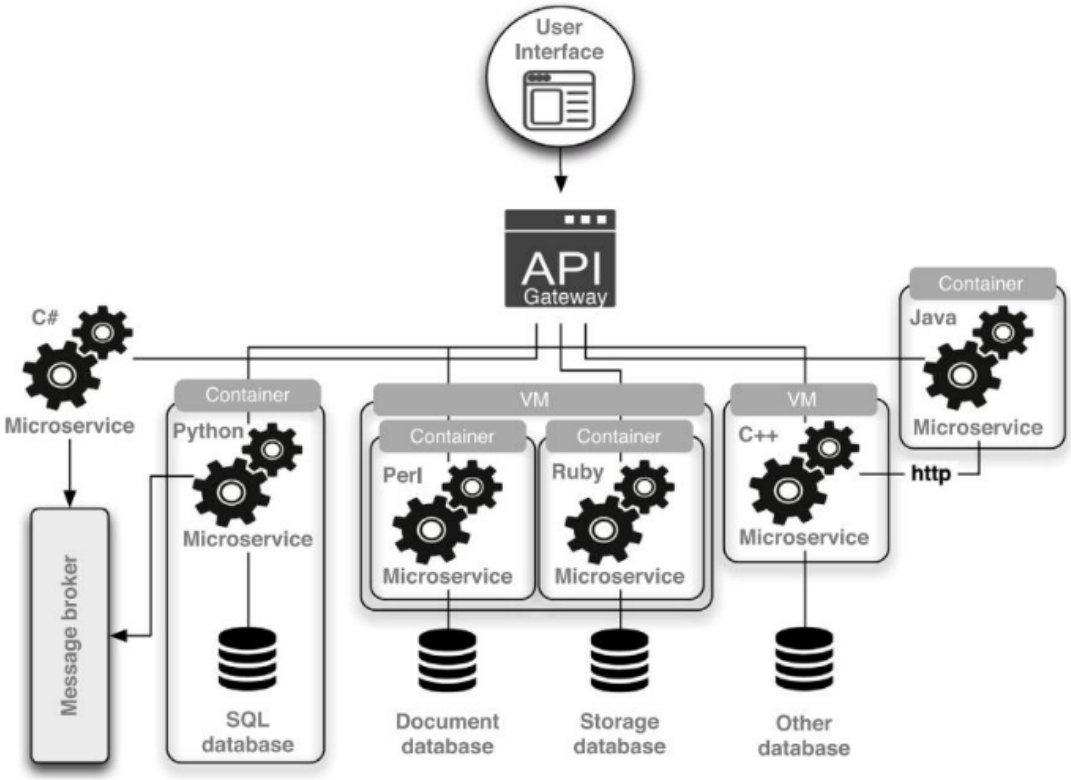
A arquitetura de microsserviços é uma solução para escalabilidade e elasticidade da infraestrutura do sistema em oposição à arquitetura monolítica (PASOMSUP; LIMPIYAKORN, 2021). A adoção da computação em nuvem e a utilização de tecnologias de contêineres impulsionam a arquitetura de microsserviços, tornando-a cada vez mais popular (HE; YANG, 2017). Em uma arquitetura de microsserviços, uma *API Gateway*

desenvolve um papel muito importante, o de controlar a entrada e a saída de requisições. A *API Gateway* direciona as requisições para os serviços apropriados, ela gerencia as respostas desses serviços e também pode traduzir ou transformar as mensagens de requisições que trafegam por ela.

O estilo arquitetônico de microsserviços oferece várias vantagens para o desenvolvimento de sistemas, mas também surgem novos desafios de segurança (PEREIRA-VALE et al., 2019). Os estudos atuais focam principalmente nas ameaças de acesso não autorizado, exposição de dados confidenciais e comprometimento de microsserviços individuais (HANNOUSSE; YAHIOUCHE, 2021). Em resumo, a arquitetura de microsserviços enfrenta o desafio de fornecer um serviço de autenticação de usuário que seja rápido, seguro e confiável (YANG et al., 2021).

A Figura 2 ilustra como os microsserviços se comunicam entre si por meio de uma *API Gateway*. Os microsserviços tem a opção de serem implantados em máquinas virtuais, plataformas de nuvem ou em contêineres. O uso de contêineres para implementação de microsserviços é preferida devido à sua simplicidade, menor custo e capacidade de inicialização e execução rápida (HANNOUSSE; YAHIOUCHE, 2021).

Figura 2 – Arquitetura de microsserviços



Fonte: Hannousse e Yahiouche (2021)

3.2 Controle de Acesso

Um projeto de segurança para um sistema deve ser concebido de forma a ser resiliente tanto a ameaças internas quanto a externas, ao mesmo tempo em que adotar padrões de segurança bem definidos é essencial para garantir uma implementação robusta com bom desempenho e escalabilidade (MELTON, 2021).

Não há um método de proteção único que atenda a todos os requisitos de segurança e especificações de um sistema distribuído (CHATTERJEE; PRINZ, 2022). A segurança em microsserviços frequentemente é subestimada, e a responsabilidade é atribuída a uma *API Gateway*, que protege os serviços internos (BHUTADA; JYOTHI, 2019).

As estratégias mais adequadas para controle de acesso na arquitetura de microsserviço são: (i) sessão distribuída, (ii) solução baseada em SSO, (iii) *JSON Web Token* (JWT) no lado do cliente e *API Gateway* (HE; YANG, 2017). Vale ressaltar que o JWT será utilizado pelo Portal de *Single Sign-On* (SSO) do SIE para autenticação de usuários.

Por meio do controle de acesso, o sistema restringe o acesso de usuários não autorizados, garantindo assim a confidencialidade e integridade de seus recursos a usuários válidos (MAJUMDER; NAMASUDRA; NATH, 2014). É um procedimento que permite, nega ou restringe o acesso do usuário a um sistema (KHAN, 2012). O controle de acesso é composto por duas partes essenciais: autenticação e autorização. Isso implica na verificação sistemática de que uma entidade que solicita o acesso a um recurso legítimo e possui os direitos necessários (PEREIRA-VALE et al., 2021).

O controle de acesso é o modo pelo qual a autenticação é concedida aos usuários com base nos direitos de acesso definidos pelo sistema. Assim, é fundamental garantir que os recursos não sejam acessados ou utilizados de forma ilegal (ZHAO; SUN, 2020). A autenticação é uma técnica utilizada para verificar a identidade, enquanto que a autorização é uma técnica usada para verificar as permissões dos usuários para acessar recursos específicos (HANNOUSSE; YAHIOUCHE, 2021).

ShuLin e JiePing (2020), recomendam utilizar um esquema de controle de acesso com base em *Open Authorization* (OAuth 2.0) para arquitetura de microsserviços. No entanto, em comparação com a arquitetura monolítica, o controle de acesso em um aplicativo de microsserviços torna-se mais complexo.

Autenticação: é um componente fundamental da segurança em qualquer sistema. Quando combinada com o mecanismo de autorização, forma uma camada de defesa essencial para proteger os diversos elementos da arquitetura de microsserviços (HANNOUSSE; YAHIOUCHE, 2021). Os protocolos de autenticação *OAuth* e *OpenID Connect*, são amplamente utilizados por grandes provedores de serviços (CHEN; HUANG; KING, 2019).

A autenticação com base em sessão é uma abordagem de segurança que mantém uma sessão entre o servidor do sistema e o navegador do usuário. Embora essa abordagem possa ser eficaz em uma aplicação monolítica, ela não é adequada para microsserviços. No contexto de microsserviços, as solicitações precisam ser roteadas para vários servi-

ços independentes, cada um em sua própria sessão. Outra abordagem de segurança é a autenticação com base em *token*. Nesse método, um *token* contendo informações criptografadas é usado para estabelecer segurança entre as partes. Por exemplo, após o login do usuário, o sistema envia um *token* criptografado ao navegador do usuário, que é incluído nas próximas solicitações para comprovar o status de login (LIU; SCHMIEDEHAUSEN; WANG, 2024).

Autorização: técnica utilizada para verificar as permissões dos usuários para acessar recursos ou informações específicas (HANNOUSSE; YAHIOUCHE, 2021). A autorização muitas vezes foi negligenciada, porém, atualmente a maioria das implementações considera a autorização como parte essencial do processo (SINGH; RAJ; SADAM, 2022).

A autorização em microsserviços é amplamente implementada por meio do protocolo OAuth 2.0, pois ele emite *tokens* de acesso para clientes confiáveis, que têm permissão para acessar serviços específicos, protegendo os microsserviços contra o acesso não autorizado (YU et al., 2019). A implementação de uma estrutura de autorização de forma correta é a base de maturidade de um software (BARABANOV; MAKRUSHIN, 2020).

SSO: é um esquema de autenticação que permite o acesso a múltiplos aplicativos e serviços utilizando apenas um conjunto de credenciais de login. Além disso, ele permite que os usuários se autentiquem uma única vez, proporcionando uma experiência mais eficiente e segura, ou seja, não há a necessidade de realizar múltiplos logins (HE; YANG, 2017).

O SSO é uma solução que permite que as contas de usuário sejam centralizadas em um único local, possibilitando que a identidade seja utilizada em diversos sistemas (MELTON, 2021). Na arquitetura de microsserviço, cada serviço pode ser visto como um aplicativo. Dessa forma, a autenticação é realizada somente uma vez, porém permite acesso a vários aplicativos (HE; YANG, 2017). Em síntese, o SSO simplifica o gerenciamento de acesso, permitindo que as equipes de Tecnologia da Informação (TI) gerenciem o acesso do usuário a várias aplicações e serviços a partir de um único local.

***Open Authorization* (OAuth 2.0)**: é um protocolo aberto definido pela RFC 6749 (ALMEIDA; CANEDO, 2022). O OAuth 2.0 é a estrutura preferida para garantir autorização segura em arquiteturas de aplicativos modernos (MCLARTY; WILSON; MORRISON, 2018).

O OAuth 2.0 é um esquema de autorização que se concentra na simplicidade e possibilita aos usuários acessarem diversos serviços sem a necessidade de compartilhar suas credenciais. Na prática, o usuário realiza autenticação em um servidor específico de autorização e obtém um *token* de acesso. Esse *token* permite o acesso por tempo limitado a recursos sem que o usuário precise se comunicar novamente com o servidor de autenticação ou fornecer suas credenciais novamente (BáNáTI et al., 2018). O OAuth 2.0 é um dos protocolos mais utilizados pelas arquiteturas de microsserviços para delegação de acesso (YU et al., 2019).

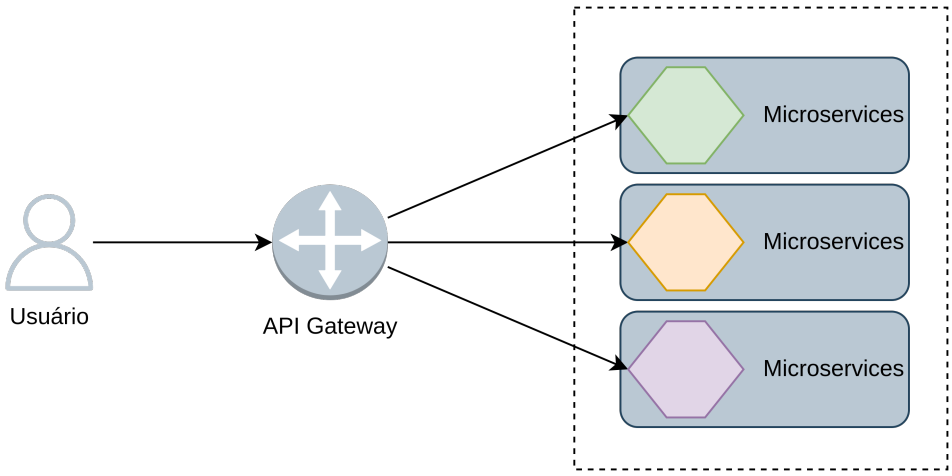
***JSON Web Token* (JWT)**: é um formato de *token* de padrão aberto largamente usado para representar informações entre partes de forma segura (BáNáTI et al., 2018). O JWT consiste em três partes principais: cabeçalho, *payload* e assinatura. O `cabeçalho` contém informações sobre o algoritmo de criptografia usado e o tipo de *token* (YANG et al., 2021). O `payload` armazena os dados, como o cabeçalho, informações sobre o usuário, permissões, expiração do *token* e outros atributos relevantes. A `assinatura` serve para garantir a integridade dos dados. Ela é gerada por meio da combinação do cabeçalho, o *payload* e uma chave secreta. A assinatura é usada para verificar se o *token* foi alterado ou adulterado durante a transmissão.

O JWT é amplamente utilizado em autenticação e autorização de API e aplicativos web, pois, ele oferece uma maneira compacta e segura de transmitir informações, sem a necessidade de armazenar dados no servidor (BáNáTI et al., 2018).

3.3 *API Gateway*

É uma camada de abstração que protege os microsserviços para não fornecer acesso direto a clientes externos. Uma *API Gateway* é posicionado em um único ponto de contato com clientes externos e ocultam com eficiência os microsserviços de *back-end* de qualquer acesso externo. Em vez de se conectar individualmente em cada microsserviço, o usuário estabelece comunicação apenas com a *API Gateway*, conforme ilustrado na Figura 3, (RAJ; VANGA; CHAUDHARY, 2023). Segundo a perspectiva deles, um dos grandes benefícios de usar uma *API Gateway* é porque ele é considerado um excelente método para reforçar os mecanismos de segurança e também pode adicionar uma série de recursos, tais como: monitoramento, roteamento, balanceamento de carga, limite de tráfego e cache.

Figura 3 – *API Gateway*



Fonte: Adaptada de Raj, Vanga e Chaudhary (2023)

Na prática, a *API Gateway* é um serviço frequentemente utilizado como intermediador, proporcionando um ponto centralizado para a implementação de políticas de

controle de acesso para microsserviços (PREUVENEERS; JOOSEN, 2019). A *API Gateway* atua entre os clientes externos e os microsserviços, fornecendo um ambiente de rede privada (XU; JIN; KIM, 2019). A *API Gateway* encapsula os detalhes de implementação do sistema e abstrai as funções comuns necessárias nos microsserviços (ZHAO; SUN, 2020). Uma *API Gateway* é um serviço que aborda sobre questões para clientes de natureza heterogêneas (MONTESI; WEBER, 2016).

Embora uma *API Gateway* ofereça diversas vantagens, sua utilização pode se tornar problemática quando ela é um único ponto de decisão, pois, em caso de falha, toda a aplicação pode ficar inacessível (BARABANOV; MAKRUSHIN, 2020).

3.4 *Log*

Logs são registros de informações, geralmente seguindo a cronologia dos eventos acontecidos e que em algum momento podem ser úteis (HOSTMIDIA, 2024). Os *logs* permitem analisar tanto o uso dos serviços quanto o comportamento dos usuários (LIU et al., 2020).

Um *Gateway* de aplicação registra os acessos em *logs* de maneira consistente, formando um sistema de proteção que inclui autenticação de usuários, controle de acesso e rastreabilidade após o acesso. Os *logs* de acesso permitem o rastreamento e a análise do comportamento dos usuários, ajudando os administradores de sistema a identificar possíveis riscos. Além disso, esses *logs* fornecem uma visão sobre o uso do sistema, oferecendo suporte para que os administradores ajustem as políticas de acesso aos serviços (FENGXUAN et al., 2023).

3.5 Trabalhos Relacionados

Para identificar trabalhos relacionados, realizamos uma busca *ad-hoc* em algumas bases científicas, sem seguir um método sistemático rigoroso. A seguir, descreveremos os trabalhos relacionados encontrados.

A interoperabilidade de serviços digitais permite aos governos ampliar o alcance de suas políticas públicas, como demonstrado pela Estônia com o *X-Road*. Este trabalho propôs uma arquitetura com a *Kong API Gateway*, oferecendo segurança similar ao *X-Road* e comparando o desempenho entre as soluções. Os resultados indicam que a arquitetura baseada na *Kong* supera o *X-Road* em termos de tempo de resposta e sugerem melhorias para ambas ferramentas (ALENCAR et al., 2022). O nosso trabalho implementa uma ferramenta de *API Gateway* e faz uma comparação de desempenho com a *Kong*. Os resultados indicam que ambas as soluções são adequadas, cada uma com suas vantagens em termos de desempenho e consumo de recursos.

Comparar o desempenho de diferentes soluções para escolher a mais adequada é um processo demorado. Para simplificar essa tarefa, o estudo apresenta o AGE, um

serviço que automatiza a implantação de diferentes cenários de *API Gateways* e fornece uma avaliação comparativa de desempenho, permitindo testes mais rápidos e eficientes em diversos ambientes (MOREIRA; RIBEIRO; SILVA, 2023). A nossa pesquisa utiliza a ferramenta *Locust* para avaliar o desempenho entre a *API Gateway* desenvolvida e a *Kong*. Realizamos uma análise comparativa considerando diferentes combinações de *endpoints*, volumes de cargas de trabalho (usuários simultâneos) e cenários de teste.

O *Anser-Gateway* é um *API Gateway* de alto desempenho em *PHP*, criado para simplificar a complexidade dos microsserviços. Com uma arquitetura baseada em *event loop* e *coroutines*, o *API Gateway* otimiza o processamento de requisições ao utilizar o escalonamento cooperativo. Comparado a outras soluções, essa abordagem oferece maior eficiência, resultando em um desempenho superior de *throughput* (LEE; TSAI, 2024). Em nosso estudo, desenvolvemos uma *API Gateway* usando o *Spring Cloud Gateway*, testamos em cenários reais e comparamos seu desempenho com a *Kong*.

O trabalho de (ZUO et al., 2020) busca otimizar o design de *API Gateways*, focando na persistência em banco de dados e na redução do acoplamento entre módulos para melhorar o desempenho, custo e manutenibilidade. A pesquisa realiza testes de desempenho para validar a eficácia da solução proposta. De forma similar, nosso trabalho também avalia o desempenho de *API Gateways*, mas com foco no gerenciamento de acessos a microsserviços, comparando diferentes soluções para determinar as vantagens de cada uma em termos de eficiência e uso de recursos.

A aplicação de *API Gateways* em sistemas embarcados com microsserviços, com foco na análise de requisitos específicos como baixo consumo de recursos, comunicação em tempo real e comparação de diferentes *API Gateways* é explorada por (TOMIć et al., 2022). Em nossa pesquisa, desenvolvemos uma *API Gateway* para aprimorar o controle de acesso aos microsserviços, comparando seu desempenho e consumo de recursos com a ferramenta *Kong*.

O projeto de uma *API Gateway* como *middleware* em uma arquitetura de *Plataforma como Serviço (PaaS)* para um sistema de serviços de laboratório é apresentado por (OKTARIA et al., 2021). A pesquisa utiliza uma abordagem orientada a serviços *(SOA)* e destaca a importância do *API Gateway* para a integração e interoperabilidade entre os microsserviços, realizando testes de desempenho para avaliar a viabilidade da solução. Nosso trabalho avalia o desempenho de *API Gateways* para gerenciar acessos a microsserviços em um sistema de gestão educacional, comparando *Kong* e a solução desenvolvida.

Por fim, a Tabela 1 contextualiza a diferença entre a abordagem dos trabalhos relacionados apresentados e a proposta desenvolvida em nossa pesquisa.

Tabela 1 – Resumo dos trabalhos relacionados

Estudo	Ferramenta Avaliada	Métrica Avaliada	Resultado	Relação com a Nossa Pesquisa
(ALENCAR et al., 2022)	Kong API Gateway vs. X-Road	Tempo de resposta e segurança	A Kong apresentou melhor tempo de resposta e sugerem melhorias para ambas ferramentas.	Comparamos a *API Gateway* desenvolvida com a *Kong*, ambas são eficientes, com vantagens específicas em desempenho e consumo de recursos.
(MOREIRA; RIBEIRO; SILVA, 2023)	AGE (Serviço Automático)	Eficiência na automatização de testes	AGE automatiza a implantação de cenários de *API Gateways* e simplifica a avaliação.	Os cenários de avaliação foram configurados na ferramenta *Locust* com base em *workloads* específicos.
(LEE; TSAI, 2024)	Anser-Gateway (PHP)	Throughput	Anser-Gateway apresentou superioridade em *throughput* devido à arquitetura baseada em event loop.	Utilizamos o *Spring Cloud Gateway* para desenvolver uma *API Gateway* e comparamos com a *Kong API Gateway*.
(ZUO et al., 2020)	Design estratégico para *API Gateway*	Desempenho e manutenibilidade	A solução proposta otimiza a persistência e reduz o acoplamento para melhorar a manutenibilidade.	Nossa *API Gateway* foca no controle de acesso e é integrada com o SIE.
(TOMIć et al., 2022)	*API Gateways* para sistemas embarcados	Consumo de recursos computacionais	O *API Gateway Tyk* atendeu à maioria dos requisitos e apresentou o melhor desempenho.	Nossa pesquisa faz uma comparação de desempenho e consumo de recursos computacionais.
(OKTARIA et al., 2021)	*API Gateway* como *Middleware* para PaaS	Eficiência no gerenciamento de requisições	Uma *API Gateway* como middleware em uma arquitetura de PaaS pode fornecer uma série de benefícios.	Apresenta uma *API Gateway* para aprimorar o controle de acesso em microsserviços institucionais.

3.6 Resumo do Capítulo

Ao longo desse capítulo apresentou-se os conceitos teóricos ligados a pesquisa deste trabalho: arquitetura de *software*, controle de acesso, *API Gateway* e *logs*, fornecendo uma base conceitual essencial para a pesquisa realizada.

A arquitetura *software* é vista como o esqueleto de um sistema, descrevendo componentes, suas funções e como se comunicam. Destacam-se dois estilos arquiteturais: a arquitetura monolítica, que concentra todos os componentes em uma única base de código, e a arquitetura de microsserviços, onde o sistema é dividido em pequenos serviços independentes, promovendo escalabilidade e flexibilidade. Destaca-se a importância de uma arquitetura bem projetada para garantir a eficiência e a manutenção do sistema.

O controle de acesso, autenticação e autorização são elementos cruciais, destacando a utilização de OAuth 2.0 e JWT para autenticação e autorização. Além disso, discute-se sobre a importância de SSO para simplificar o gerenciamento de acessos.

Uma *API Gateway* ajuda a proteger os microsserviços de acessos externos. Uma *API Gateway* centraliza a comunicação com clientes externos, ocultando os serviços de *back-end* e proporcionando segurança. Ele também pode oferecer funcionalidades como monitoramento, balanceamento de carga e cache, reforçando a segurança e eficiência do sistema.

Descrevemos como os *logs* desempenham um papel fundamental na administração

de sistemas, registrando informações importantes em ordem cronológica que podem ser úteis para diversas análises. Um *Gateway* de aplicação, ao registrar consistentemente os acessos, cria um sistema de proteção robusto com autenticação de usuários, controle de acesso e também possibilita o rastreamento e a análise do comportamento dos usuários. Isso ajuda os administradores a identificar riscos potenciais e a ajustar as políticas de acesso, assegurando um gerenciamento mais eficiente do sistema.

Para finalizar o capítulo, analisamos diferentes estudos que investigam soluções, ferramentas e tecnologias para o controle de acesso, bem como mecanismos de segurança utilizados.

4 REVISÃO SISTEMÁTICA DA LITERATURA

A migração de sistemas atualmente se concentra na modernização de sistemas legados e monolíticos para arquiteturas de microsserviços. O modelo de arquitetura de microsserviços divide uma aplicação em pequenos serviços independentes e escaláveis. A arquitetura de microsserviços pode trazer benefícios, porém, também traz desafios, principalmente quanto ao aspecto de segurança para autenticação de usuários. Para enfrentar esses desafios, soluções como API *gateways* e *proxies* são utilizadas para fazer o controle de acesso do usuário a um conjunto de aplicações que rodam sob microsserviços. Neste trabalho buscamos encontrar estudos que abordam aspectos sobre soluções, ferramentas e tecnologias sobre autenticação em arquitetura de microsserviços. A busca sistemática resultou em 3.299 estudos de pesquisa potencialmente relevantes e aplicamos um procedimento de seleção resultando em 19 trabalhos de pesquisa selecionados nos últimos 8 anos. Os resultados da revisão revelaram a identificação de 5 ferramentas de autenticação mais mencionadas pelos estudos: “Keycloak” foi a mais abordada seguida por, “Zuul”, “Kong”, “OKTA” e “Apache APISIX” seguidas de outras ferramentas que foram citadas apenas uma única fez.

A abordagem arquitetural de microsserviços tornou-se cada vez mais popular, devido à sua flexibilidade e por ser composta por vários pequenos serviços independentes, cada um executando em seu próprio processo e se comunicando por meio de protocolos leves (ALMEIDA; CANEDO, 2022). A arquitetura de microsserviços reduz o acoplamento entre os módulos do aplicativo e é benéfica para o desenvolvimento e manutenção dos sistemas. Ela fornece uma solução alternativa e flexível para escalabilidade de infraestrutura, diferentemente da arquitetura monolítica (ALSHUQAYRAN; ALI; EVANS, 2016).

A medida que se começa a transição de um sistema monolítico para microsserviços, surge o problema de autenticação e autorização para arquitetura de microsserviços. Autenticação e autorização em um sistema monolítico é diferente da arquitetura de microsserviços. Na monolítica, a autenticação e a autorização são realizadas apenas uma vez, pois nesta arquitetura o processo de aplicação não é separado. Já na de microsserviços, cada processo da aplicação pode ser tratado de forma independente de acordo com cada serviço, *e.g.* um microsserviço para tratar solicitações de autenticação e outro para autorização, e assim por diante (TRIARTONO; NEGARA; SUSSI, 2019). Já Alshuqayran, Ali e Evans (2016) define que o estilo arquitetônico de microsserviços enfatiza a divisão do sistema em vários serviços independentes, pequenos e leves.

Aspectos de autenticação e autorização são relevantes independentemente da arquitetura utilizada, pois são considerados componentes-chave para os mecanismos de segurança da aplicação (ALMEIDA; CANEDO, 2022).

Um API *gateway* é responsável por tarefas de autenticação e roteamento que atua como um único ponto de entrada para os clientes dos microsserviços do sistema. Em vez de

chamar os microsserviços diretamente, os clientes chamam o API *gateway*, o qual recebe e redireciona a solicitação para o microsserviço apropriado. Quando o microsserviço responde à solicitação, o API *gateway* a retorna ao cliente (RAJ; VANGA; CHAUDHARY, 2023). A autenticação é um mecanismo no qual o usuário com credenciais validas pode ter acesso ao serviço. A autenticação responde a questão: Quem é você? A autorização, verifica se o usuário autenticado tem permissão para consumir ou não determinada informação ou serviço. Autorização responde à seguinte pergunta: Que permissões o usuário tem? Em resumo, a arquitetura de microsserviços enfrenta o desafio de segurança para autenticação e autorização do usuário de forma centralizada (RAJ; VANGA; CHAUDHARY, 2023).

A pesquisa foi conduzida usando quatro bibliotecas digitais amplamente usadas. Aplicamos um protocolo rigoroso para extrair, classificar e organizar os estudos, resultando na remoção de 474 estudos duplicados e na rejeição de 2.806 estudos. Selecionamos cuidadosamente 19 estudos primários publicados de 2015 a janeiro de 2023, de um total de 3.299 estudos encontrados na literatura. Nesta Revisão Sistemática da Literatura (RSL) pretendemos encontrar recomendações abrangentes que abordem ferramentas, tecnologias e desafios relacionados a autenticação e autorização em arquitetura de microsserviços. Acreditamos que os resultados do nosso estudo podem trazer benefícios tanto para pesquisadores quanto para profissionais interessados em autenticação e autorização em arquitetura de microsserviços.

4.1 Protocolo da Revisão

Nesta seção, apresentamos os detalhes do protocolo adotado para a realização desta revisão sistemática da literatura. Para atingir o objetivo da pesquisa, realizamos uma coleta sistemática da literatura existente e, para auxiliar na triagem e análise dos estudos encontrados, utilizamos a ferramenta online Thoth[1].

4.1.1 Escopo e Objetivo

O principal objetivo desta RSL é identificar estudos publicados nas principais bibliotecas digitais da literatura no período de janeiro de 2015 a janeiro de 2023 que discutam sobre soluções de autenticação e autorização para arquitetura de microsserviços.

4.1.2 Questões de Pesquisa

Definimos as seguintes Questões de Pesquisa (QP):

QP1. Quais são as ferramentas de gerenciamento de autenticação existentes para arquitetura de microsserviços na literatura?

[1] Thoth: <http://200.132.136.13/Thoth/>

(Microservice OR Microservices OR Container OR "Distributed Application") AND (Authentication OR Authenticate OR Authorization OR "API Gateway" OR Access OR Proxy) AND (Management OR Identity OR Control OR Permission OR Security) AND (Approach OR Mechanism* OR Framework OR Architecture OR Strateg* OR Protocol* OR Solution*)*

Figura 4 – *String* de busca genérica.

QP2. Quais são as tecnologias mais utilizadas pelas ferramentas de autenticação para arquitetura de microsserviços?

QP3. Quais são os desafios relacionados aos aspectos de autenticação de sistemas na arquitetura de microsserviços?

4.1.3 Processo de Pesquisa

O método aplicado nesta pesquisa é o processo clássico de três etapas para a execução de estudos sistemáticos: Planejamento, Condução e Relatório proposto por Kitchenham e Brereton (2013). Para a realização da pesquisa foram utilizadas bibliotecas digitais que atendem aos seguintes critérios (1) possuem um mecanismo de pesquisa baseado na web; (2) tem um mecanismo de busca capaz de usar palavras-chave, e; (3) conter trabalhos da área de ciência da computação. Realizamos a busca nas principais bibliotecas digitais na área de Computação. As bibliotecas utilizadas foram IEEE Xplore, ACM Digital Library, Scopus e Engineering Village, todas acessadas em 12 de janeiro de 2023. Na Figura 4 apresentamos a *string* de busca utilizada, que foi definida de acordo com as palavras-chave que devem aparecer nos resultados da busca. Para formular a *string* de busca, utilizamos termos e sinônimos, as operações booleana "OR" para selecionar sinônimos alternativos para cada termo e "AND" para combinar os termos. Maiores detalhes sobre a execução da *string* em cada biblioteca digital pode ser encontrado em arquivos de imagens complementares disponibilizados no repositório Zenodo da RSL[2].

4.1.4 Critérios de Inclusão e Exclusão

Os critérios de seleção buscam identificar estudos que forneçam informações sobre as QPs. Definimos os seguintes **Critérios de Inclusão (CI)**:

CI1. *Trabalhos publicados entre os anos de 2015 a 2023;*

CI2. *Estudos sobre soluções de gerenciamento de autenticação em arquitetura de microsserviços;*

CI3. *Publicações relacionadas a ferramentas de autenticação para arquitetura escalável.*

Critérios de Exclusão (CE):

CE1. *Trabalhos que não estão escritos no idioma inglês;*

[2] Repositório Zenodo: <https://zenodo.org/records/10161123>

CE2. *Arquivo PDF não disponível;*

CE3. *Publicações que não atendam os critérios de inclusão.*

4.1.5 Critérios de Qualidade

O objetivo da avaliação de qualidade é avaliar os estudos, como forma de mensurar sua relevância em relação a outros. Os Critérios de Qualidade (CQ) foram baseados em (DYBa; DINGSØYR, 2008): relevância (CQ1, CQ2 e CQ3), relatório (CQ4), rigor (CQ5) e credibilidade (CQ6). Cada um dos critérios de qualidade são avaliados de acordo com a seguinte nota: S (sim) = 100%; P (parcial) = 50%, N (não) = 0%.

CQ1. *O estudo apresenta soluções de gerenciamento de autenticação para arquitetura de microsserviços? Avaliação*: **S**: Estudo apresenta pelo menos uma solução de autenticação; **P**: Estudo apresenta em partes uma solução de autenticação; **N**: Estudo não apresenta uma solução de autenticação;

CQ2. *O estudo apresenta ferramentas de autenticação para se utilizar em uma arquitetura escalável? Avaliação*: **S**: O estudo apresenta pelo menos uma ferramentas autenticação; **P**: O estudo apresenta em partes uma ferramentas autenticação; **N**: Estudo não apresenta uma ferramenta de autenticação;

CQ3. *O estudo identifica problemas e/ou desafios envolvendo autenticação na arquitetura de microsserviços? Avaliação*: **S**: O estudo identifica problemas e/ou desafios na autenticação em arquitetura de microsserviços; **P**: O estudo identifica em partes problemas e/ou desafios na autenticação em arquitetura de microsserviços; **N**: O estudo não identifica problemas e/ou desafios na autenticação em arquitetura de microsserviços;

CQ4. *O objetivo da pesquisa realizada está claramente descrito? Avaliação*: **S**: O objetivo está claramente descrito; **P**: O objetivo está parcialmente descrito; **N**: O objetivo não está claramente descrito;

CQ5. *A análise dos dados foi suficiente para abordar os objetivos da pesquisa? Avaliação*: **S**: A análise dos dados foi suficiente para abordar os objetivos da pesquisa; **P**: A análise dos dados foi parcialmente suficiente para abordar os objetivos da pesquisa; **N**: A análise dos dados não foi suficiente para abordar os objetivos da pesquisa;

CQ6. *Existe uma declaração clara das conclusões? Avaliação*: **S**: Existe uma declaração clara das conclusões; **P**: Existe parcialmente uma declaração clara das conclusões; **N**: Não existe uma declaração clara das conclusões.

4.1.6 Processo de Seleção

Esta seção apresenta as etapas realizadas no processo de busca e seleção dos estudos:

(i) Execução da *string* de busca nas bibliotecas digitais;

(ii) Exportação dos arquivos no formato BibTeX com os estudos e importação dos mesmos na ferramenta Thoth;

(iii) Remoção dos estudos primários duplicados;

(iv) Aplicação dos critérios de inclusão e exclusão com base na leitura do título e o resumo;

(v) Avaliação de qualidade e extração de dados com a leitura na íntegra dos estudos;

(vi) Extração de dados dos estudos primários selecionados;

(vii) Análise qualitativa com apoio da ferramenta QAnubis dos dados extraídos.

A Figura 5 mostra a execução do processo seleção dos estudos, com a respectiva quantidade identificada em cada etapa do protocolo.

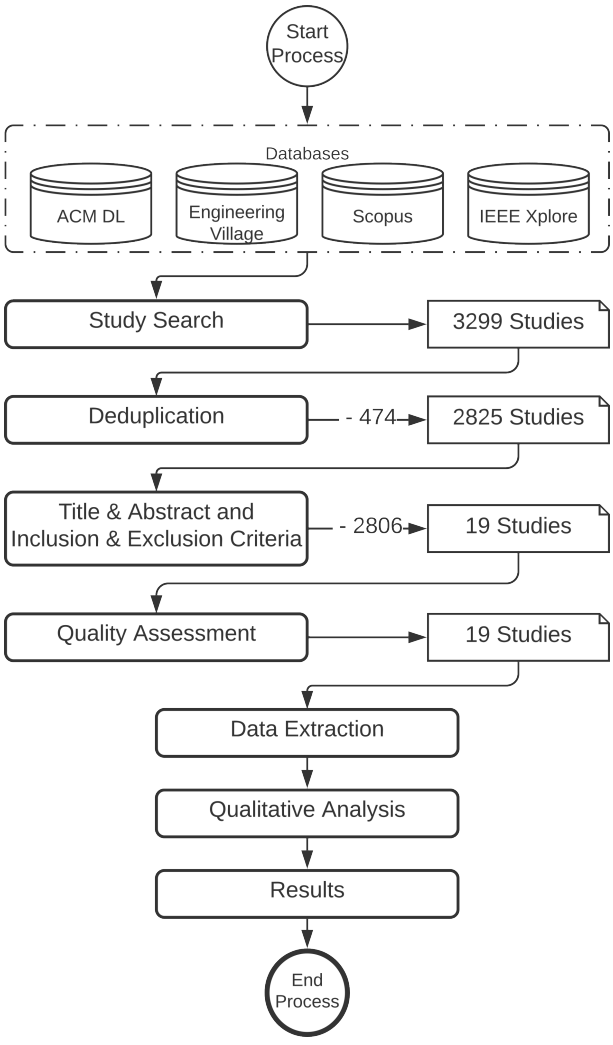


Figura 5 – Processo de seleção dos estudos.

4.1.7 Extração de Dados

Para realizar a extração dos dados, aplicamos um formulário utilizando a ferramenta Thoth com o objetivo de extrair informações relevantes para responder às nossas questões de pesquisa. Lemos atentamente cada um dos estudos e preenchemos o formu-

Tabela 2 – Estudos selecionados em cada biblioteca digital.

Biblioteca	Total	Duplicados	Rejeitados	Aceitos
ACM DL	119	3	115	1
IEEE Xplore	234	6	218	10
Engineering Village	1.216	211	1.000	5
Scopus	1.730	254	1.473	3
	3.299	**474**	**2806**	**19**

lário. Assim, registramos os dados extraídos de cada estudo em uma planilha (Google Planilhas) para categorizar, comparar e melhor identificar as soluções de autenticação e autorização utilizadas na arquitetura de microsserviços.

Coletamos as seguintes informações de cada estudo: título, resumo, autores, ano de publicação, base de dados, desafios da solução, ferramentas, detalhes da solução e a classificação da solução. A extração de dados referentes às ferramentas citadas na literatura para tratar da autenticação no contexto da arquitetura de microsserviços está diretamente relacionada à intenção de abordar a QP1. Com o objetivo de buscar respostas para a QP2, coletamos informações referentes aos detalhes da solução apresentados nos estudos. Para abordar a QP3, agregamos dados relacionados aos desafios mencionados pelos estudos selecionados.

4.2 Condução

Esta seção explica como conduzimos nossa RSL.

4.2.1 Bibliotecas Digitais

Apresentamos uma análise da execução da *string* de busca em cada uma das bibliotecas digitais (BD). Limitamos a busca aos campos "Resumo" e "Título" em todas as BDs, excluindo o "Corpo" do estudo. As BDs IEEE Xplore e Engineering Village permitem refinar a busca, determinando o intervalo de ano da publicação direto na interface da aplicação. Já as BDs ACM Digital Library e Scopus o filtro foi incorporado como parte da *string* de busca. A aplicação da *string* em cada uma das BDs, totalizou 3299 estudos e 474 duplicados foram removidos. Isso reduziu a quantidade de estudos para 2825. Ao examinar o título e o resumo dos estudos restantes, 2806 foram rejeitados por sua irrelevância. Restando 19 estudos aprovados que tem alguma relevância com os critérios de inclusão. A Tabela 2 resume os quantitativos dos estudos em cada banco de dados.

4.2.2 Avaliação de Qualidade

Na etapa de avaliação de qualidade, aplicamos os CQs (Seção 4.1.5) para avaliar a confiabilidade e ranquear os melhores estudos. Como critério de exclusão na avaliação de qualidade, o estudo teria que ser pontuado como Fraco ou Médio, como nenhum dos

estudos foi definido com essa classificação, manteve-se os 19 estudos para o andamento deste trabalho.

A Tabela 3 fornece informações sobre a pontuação de cada estudo após o processo de CQ. A coluna **Referência** identifica cada estudo. As colunas 2 a 7 exibem as pontuações com base nas perguntas de controle de qualidade. A penúltima coluna **C** exibe o conceito de cada estudo e última coluna **P** mostra a pontuação final de cada estudo.

Tabela 3 – Critério de Qualidade.

Referência	QA1	QA2	QA3	QA4	QA5	QA6	Conceito	P
Bhutada e Jyothi (2019)	●	○	◐	●	◐	●	Bom	4
Bánáti et al. (2018)	●	◐	●	●	◐	●	Excelente	5
Chatterjee e Prinz (2022)	●	●	◐	◐	●	●	Excelente	5
Das et al. (2021)	●	●	◐	◐	◐	◐	Bom	4
He e Yang (2017)	●	○	●	●	◐	◐	Bom	4
Liu et al. (2020)	◐	◐	◐	●	◐	◐	Bom	3.5
Melton (2021)	●	●	●	◐	◐	◐	Bom	4.5
Pasomsup e Limpiyakorn (2021)	●	●	●	◐	◐	◐	Bom	4.5
Pontarolli et al. (2021)	●	○	●	◐	◐	◐	Bom	3.5
Preuveneers e Joosen (2019)	●	●	●	◐	●	◐	Excelente	5
Raj, Vanga e Chaudhary (2023)	●	●	●	◐	◐	◐	Bom	4.5
Ranawaka et al. (2020)	●	●	●	●	◐	◐	Excelente	5
ShuLin e JiePing (2020)	●	●	●	◐	◐	●	Excelente	5
Snger e Abeck (2022)	●	◐	◐	◐	◐	●	Bom	4
Triartono, Negara e Sussi (2019)	●	◐	◐	◐	◐	●	Bom	4
Xiong e Li (2022)	●	●	●	●	◐	●	Excelente	5.5
Xu, Jin e Kim (2019)	●	●	◐	●	◐	●	Excelente	5
Yang et al. (2021)	●	●	●	◐	◐	◐	Bom	4.5
Yarygina e Bagge (2018)	●	●	◐	◐	◐	◐	Bom	4

Legenda: **Sim** = ●| **Parcial** = ◐| **Não** = ○| **P** = Pontuação

4.3 Análise dos Resultados

Nesta seção, apresentamos os resultados obtidos em nossa RSL em resposta às QPs.

4.3.1 Respostas às Questões de Pesquisa

QP1. *Quais são as ferramentas de gerenciamento de autenticação para uma arquitetura de microsserviços existentes na literatura?* Entre os 19 estudos primários analisados, Keycloak é a ferramenta mais mencionada, seguida por Zuul, Kong, Okta e Apache APISIX. A popularidade da Keycloak provavelmente se deve ao fato de ser uma ferramenta *open-source*, o que facilita sua adoção por diversas organizações, além de sua capacidade de integração com protocolos como *OAuth2* e *OpenID Connect*. Ela também oferece uma arquitetura escalável, e a funcionalidade de *Single Sign-On (SSO)* simplifica o gerenciamento centralizado de identidades. Na Tabela 4 apresentamos as ferramentas que implementam autenticação e autorização identificadas na literatura. Classificamos 12 ferramentas e fornecemos uma breve descrição de cada uma delas.

Tabela 4 – Soluções encontradas nos estudos primários selecionados.

Soluções	**Tipo**	**Link**	**Estudos**
Apache APISIX	F	<apisix.apache.org>	(RAJ; VANGA; CHAUDHARY, 2023)
Dubbo	F	<dubbo.apache.org>	(YANG et al., 2021)
Express Gateway	F	<www.express-gateway.io>	(RAJ; VANGA; CHAUDHARY, 2023)
Gloo	F	<docs.solo.io/gloo-edge>	(RAJ; VANGA; CHAUDHARY, 2023)
Goku	F	<www.gokuapi.com>	(RAJ; VANGA; CHAUDHARY, 2023)
Keycloak	F	<www.keycloak.org>	(RANAWAKA et al., 2020; MELTON, 2021; PREUVENEERS; JOOSEN, 2019; CHATTERJEE; PRINZ, 2022; DAS et al., 2021)
Kong	F	<konghq.com>	(RAJ; VANGA; CHAUDHARY, 2023; XU; JIN; KIM, 2019)
KrakenD	F	<www.krakend.io>	(RAJ; VANGA; CHAUDHARY, 2023)
Ocelot	F	<ocelot.readthedocs.io>	(RAJ; VANGA; CHAUDHARY, 2023)
Okta	F	<www.okta.com/>	(SNGER; ABECK, 2022)
Tyk	F	<tyk.io>	(RAJ; VANGA; CHAUDHARY, 2023)
Zuul	F	<github.com/Netflix/zuul>	(SHULIN; JIEPING, 2020; XIONG; LI, 2022)
OpenID Connect	P	<openid.net/connect>	(BáNáTI et al., 2018; MELTON, 2021; SNGER; ABECK, 2022; CHATTERJEE; PRINZ, 2022; YARYGINA; BAGGE, 2018)
OAuth	P	<oauth.net/2/>	(TRIARTONO; NEGARA; SUSSI, 2019; BáNáTI et al., 2018; SHULIN; JIEPING, 2020; PASOMSUP; LIMPIYAKORN, 2021; SNGER; ABECK, 2022; CHATTERJEE; PRINZ, 2022; YARYGINA; BAGGE, 2018; LIU et al., 2020; YANG et al., 2021; MELTON, 2021)
JWT	S	<jwt.io>	(PASOMSUP; LIMPIYAKORN, 2021; BáNáTI et al., 2018; SHULIN; JIEPING, 2020; MELTON, 2021; SNGER; ABECK, 2022; XU; JIN; KIM, 2019; YARYGINA; BAGGE, 2018; BHUTADA; JYOTHI, 2019; HE; YANG, 2017; PONTAROLLI et al., 2021; RAJ; VANGA; CHAUDHARY, 2023; PREUVENEERS; JOOSEN, 2019; YANG et al., 2021)

Legenda: **F** = Ferramenta | **P** = Protocolo | **S** = *Standard*.

Apache APISIX é um API *gateway stateless* projetado para trabalhar com um grande número de solicitações e com alto desempenho, é leve, nativo da nuvem e fácil de conteinerizar. Suporta uma ampla gama de protocolos e pode se conectar com serviços como Auth0 e Okta (RAJ; VANGA; CHAUDHARY, 2023).

Dubbo é uma estrutura distribuída de alto desempenho de código aberto do Alibaba (YANG et al., 2021). Dentre muitas outras funcionalidades, como descoberta de serviço e roteamento, Dubbo fornece um mecanismo de controle de acesso para aplicativo em uma interface para autenticar as solicitações dos clientes (CLOUD, 2023).

Express Gateway é um API *gateway* de microsserviços simples, rápido, flexível e com uma enorme gama de recursos. Possui um módulo de gerenciamento de credenciais que fornece um sistema completo de autenticação e autorização (GATEWAY, 2023).

Gloo é um API *gateway* com recurso de roteamento em nível de função e com suporte a várias formas de autenticação. É projetado para oferecer suporte a aplicativos com várias tecnologias, arquiteturas, protocolos e nuvens podem coexistir (EDGE, 2023).

Goku é um *gateway* de API projetado para arquitetura de microsserviços com excelente desempenho. Fornece gerenciamento de tráfego, conversão de protocolo, roteamento dinâmico, balanceamento de carga, descoberta de serviço, autenticação de identidade, e outras funções (TECHNOLOGY, 2023).

Keycloak é um provedor de autenticação e autorização que atua como um único ponto de entrada para um conjunto de microsserviços (MELTON, 2021). Para (CHATTERJEE; PRINZ, 2022), Keycloak é uma plataforma de acesso aberto que protege aplicações Web, serviços *REST-Full* e APIs utilizando protocolos como *OAuth* e *OpenID*.

Kong é um *API Gateway* escalável, de alto desempenho, nativo da nuvem e fornece recursos de roteamento, autenticação, balanceamento de carga entre muitos outros (KONG, 2023). A *Kong API Gateway* funciona como um ponto de entrada central para orquestrar o tráfego de microsserviços (ALENCAR et al., 2022). A Kong atua na frente de qualquer tipo de serviço e pode ser estendido por meio de *plug-ins*, oferecendo funcionalidades adicionais, como autenticação via *JSON Web Token* (JWT), Controle de Tráfego, *Logging*, entre outros (XU; JIN; KIM, 2019).

KrakenD possui como principal funcionalidade a criação de uma API que atua como um agregador de muitos microsserviços em endpoints únicos, fazendo o trabalho pesado automaticamente: rotear, agregar, transformar, filtrar, autenticar e muito mais. KrakenD é um gateway de camada 7 puro, não acoplado à camada de transporte HTTP (KRAKEND, 2023).

Ocelot destina-se a aplicações que usam o *framework* .NET e são executadas em arquitetura de microsserviços que precisam de um ponto de entrada unificado em seu sistema. A principal funcionalidade do Ocelot é receber solicitações e encaminhá-las para o serviço de destino, mas não se limita a isso, possui muitos outros recursos como: autenticação, autorização, balanceamento de carga, descoberta de serviço entre muitos outros (PALLISTER, 2023).

Okta fornece uma solução de gestão de identidade e acesso como serviço para registrar e autenticar usuários. Okta atua como um provedor de identidade com foco na segurança e integrando mecanismos de autenticação e autorização usando os protocolos OIDC e OAuth (SNGER; ABECK, 2022).

Tyk oferece um API *Gateway* rápido, escalável e moderno, compatível com vários protocolos, formas de autenticação e controle de acesso granular (TECHNOLOGIES, 2023).

Zuul é construído para realizar funções como roteamento dinâmico, segurança, autenticação e monitoramento (XIONG; LI, 2022). De acordo com (YANG et al., 2021), quando os usuários precisam acessar os microsserviços por meio do Zuul, eles precisam fazer login pela primeira vez e, em seguida, ir para o servidor de autorização OAuth para autenticação e autorização. Atualmente, o *Zuul* é um projeto descontinuado que não recebe mais atualizações nem *patches*. Inspirado no *Zuul*, foi desenvolvido o *Spring Cloud Gateway* como seu sucessor no ecossistema de projetos do *Spring Boot* (SANCHEZ; CARNELL, 2021).

QP2. *Quais são as tecnologias mais utilizadas pelos mecanismos de autenticação para arquitetura de microsserviços?* As principais tecnologias utilizadas para autentica-

Resumo da QP1. Dentre as ferramentas mencionadas pelos estudos sobre autenticação e autorização para arquitetura de microsserviços, a *keycloak* se destacou como a mais citada (58 menções), seguida por *Zuul* (17 menções), *Kong* (14 menções), *Okta* (13 menções), *Apache APISIX* (4 menções) e outras 7 ferramentas que foram mencionadas apenas um vez.

ção e autorização identificadas na literatura são apresentadas na Tabela 4, destacadas por **P**rotocolos e um padrão (**S**tandard). Entre os estudos selecionados, destacam-se os protocolos *OAuth* e *OpenID Connect*, bem como o mecanismo *JWT*, como as tecnologias mais citadas.

***OAuth* (*Open Authorization*)** é um protocolo definido pela RFC 6749 e sua estrutura de autorização concentra-se na simplicidade e permite autenticação em diversos serviços sem ter que compartilhar as credenciais. OAuth é um dos protocolos mais utilizados pelas arquiteturas de microsserviços para autenticação quanto de autorização (ALMEIDA; CANEDO, 2022).

OpenID Connect é um protocolo aberto de identidade que utiliza os mecanismos do *OAuth* (YU et al., 2019). Ele garante que com apenas uma identidade digital é possível autenticar em vários aplicativos (ALMEIDA; CANEDO, 2022).

JWT (*JSON Web Token*) é um padrão aberto definido pela RFC 7519, que estabelece uma maneira compacta e independente de transmitir informações com segurança entre o servidor e o cliente (YANG et al., 2021).

Resumo da QP2. Os protocolos *OAuth*, *OpenID Connect* e o mecanismo JWT foram as principais tecnologias identificadas pelos estudos para auxiliar na implementação de uma ferramenta de autenticação para arquitetura de microsserviços.

QP3. *Quais os principais desafios relacionados aos aspectos de autenticação de sistemas na arquitetura de microsserviços?* Os desafios mais relevantes identificados nos estudos sobre autenticação de sistemas na arquitetura de microsserviços foram citados 41 vezes nos estudos. Os desafios mais frequentemente mencionados na literatura foram: [label=()]

Segurança/Privacidade (13 menções),

Autenticação e Autorização/Controle de Acesso (9 menções),

Arquitetura de Microsserviços (7 menções), e

Aumento da superfície/grande número de microsserviços (4 menções). Entre os desafios, observamos a sua interligação direta, pois todos estão relacionados com mecanismos de segurança. Alguns estudos que abordaram esses desafios forneceram uma explicação da estrutura monolítica tradicional e da arquitetura de microsserviços. Isso ocorre porque a

autenticação em microsserviços difere da abordagem monolítica e, portanto, é um ponto desafiador para a segurança na arquitetura de microsserviços.

Segurança/Privacidade: A segurança é um dos maiores desafios na construção de sistemas do mundo real, portanto, há uma necessidade urgente de abordar as questões de segurança na arquitetura de microsserviços (YARYGINA; BAGGE, 2018). Apesar dos benefícios da computação distribuída, ela pode apresentar diversos desafios relacionados à segurança e privacidade (XU; JIN; KIM, 2019).

Autenticação e Autorização/Controle de Acesso: O aumento no número de serviços resulta em um desafio para manter a segurança no controle de acesso. Com mais superfícies, os riscos à segurança e à privacidade também se multiplicam (PASOMSUP; LIMPIYAKORN, 2021). Para mitigar os desafios ao controle de acesso em ambientes de microsserviços são apresentadas as soluções como JWT, OAuth 2.0, *Role-Based Access Control* (RBAC), SSO, *OpenID Connect* e *API Gateway* (ARAUJO; MARINHO, 2023).

"A autorização é implementada usando *tokens* JWT, permitindo a aplicação do controle de acesso no *Gateway* da API para remover essas preocupações dos microsserviços, de modo que os microsserviços possam permanecer leves para atender aos requisitos de dispositivos de borda com recursos limitados (XU; JIN; KIM, 2019)".

Arquitetura de Microsserviços: Ranawaka et al. (2020) citam que "Embora a arquitetura baseada em microsserviços do Custos aproveite as melhores práticas para a construção de tais sistemas, enfrentamos vários desafios arquitetônicos seguindo esta abordagem". O Custos é uma estrutura de software que fornece operações de segurança para *gateways* científicos. Para Yang et al. (2021) "...a arquitetura de microsserviços enfrenta o desafio da autenticação rápida do usuário".

Aumento da superfície/grande número de microsserviços: Pasomsup e Limpiyakorn (2021) explicam que "... os desafios de segurança dos microsserviços são a crescente superfície de ataque das aplicações e a vulnerabilidade desconhecida". Pontarolli et al. (2021) relatam que "Os Microsserviços apresentam problemas de segurança desafiadores, e pelo grande número de microsserviços aumenta muito a dificuldade de monitorar a segurança de toda a aplicação".

Resumo da QP3. Os desafios identificados relacionados aos microsserviços incluem (i) Segurança/Privacidade, (ii) Autenticação e Autorização/Controle de Acesso, (iii) Arquitetura de Microsserviços, e (iv) Aumento da superfície/grande número de microsserviços.

4.3.2 Análise Qualitativa

Para análise qualitativa, utilizamos a ferramenta QAnubis desenvolvida pelo nosso grupo de pesquisa. A Tabela 5 apresenta um resumo das grandezas relevantes relacionadas

à análise temática realizada. Realizamos citações em forma de codificação em palavras, frases ou parágrafos que contém uma ideia clara que pode ser atribuída a um código específico. Durante a análise dos estudos, identificamos um total de 1019 citações codificadas em 8 temas. Uma citação codificada consiste em um trecho de texto (palavras, frases ou parágrafos) extraído dos estudos analisados, que expressa uma ideia clara. Essas ideias são associadas a códigos específicos, representando temas relevantes. Durante a análise, as citações codificadas são organizadas e agrupadas em temas abrangentes, proporcionando uma visão estruturada e coerente dos tópicos abordados nos estudos.

Tabela 5 – Estatísticas relacionadas aos códigos e cotações.

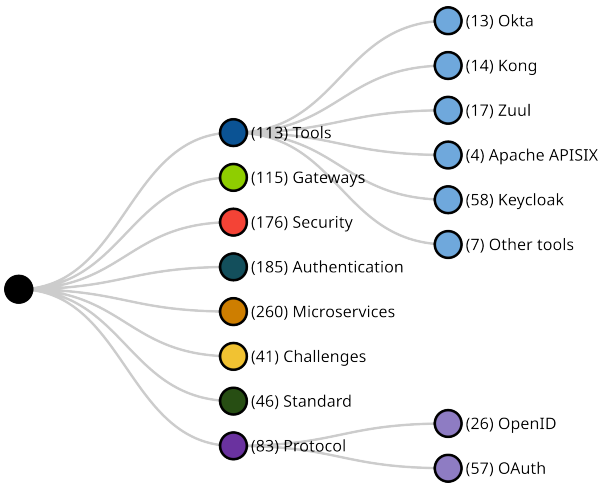
# Número de citações codificadas	1019
# Número de citações	16
# Número de estudos	19
μ Média de citações por código	63,68
μ Média de citações por estudo	53,63

Dentre os temas, Microsserviços, Autenticação e Segurança se destacam, recebendo 260, 185 e 176 menções respectivamente, citados em todos os 19 estudos selecionados. A Figura 6(a) apresenta os códigos identificados e a árvore de temas. A predominância desses temas ressalta a importância desses conceitos no contexto dos estudos analisados. Esses temas estão diretamente relacionados às questões de controle de acesso, elemento essencial para ambientes de microsserviços. A alta ocorrência desses temas aponta que são assuntos recorrentes nas discussões sobre as práticas de microsserviços.

Apresentamos uma nuvem de palavras que representa as mais citadas nos estudos analisadas. A nuvem de palavra exibida na Figura 6(b) incorpora uma Folksonomia (XU et al., 2008), onde termos notáveis como “autenticação”, “segurança”, “microsserviço(s)”, “gateway”, “API”, entre outros, são exibidos. A nuvem de palavras serve como uma representação visual que destaca os termos mais frequentes, refletindo as principais preocupações e prioridades dos autores nos estudos analisados. A predominância de termos relacionados à arquitetura, como “microsserviços”, “API” e “gateway” ressaltam a importância do controle de acesso, que é um pilar para a segurança e integração dos microsserviços na construção de sistemas modernos. A análise de uma nuvem de palavras pode revelar tendências sobre a pesquisa e desenvolvimento de soluções de autenticação para microsserviços. Elaboramos algumas possíveis conclusões que podem ser extraídas da nuvem de palavras:

A *folksonomia* pode revelar termos que estão começando a se destacar na literatura, mas ainda não são amplamente discutidos. Esses termos podem indicar novas áreas de pesquisa ou tecnologias em ascensão. A coocorrência de certos termos podem indicar relações temáticas ou áreas de interseção dentro dos estudos analisados. A presença conjunta de termos como “segurança”, “autenticação”, “microsserviços”, “API’ e “Gateway”

pode sugerir uma ênfase em controle de acesso na arquitetura de software baseada em microsserviços. Ao analisar a frequência relativa dos termos, é possível deduzir quais tópicos recebem mais atenção dos pesquisadores. Se termos como "segurança", "autenticação", "microsserviços", "API" e "Gateway" aparecem com destaque, isso indica que esses são os principais focos dos estudos, refletindo as preocupações atuais da comunidade científica. A baixa frequência de certos termos que poderiam ser esperados em um contexto específico pode revelar lacunas na pesquisa. Por exemplo, se termos como "autorização" ou "gerenciamento" aparecem com pouca frequência, isso pode indicar uma área que ainda necessita de maior investigação. Termos relacionados a tecnologias específicas, como "API", "Gateway" e "Microsserviços", podem indicar quais ferramentas e tecnologias são mais influentes nas soluções discutidas nos estudos.



((a)) Árvore de Códigos e Temas.

((b)) Folksonomy (XU et al., 2008) - Nuvem de Palavras das Citações.

Figura 6 – Resultados da Análise Qualitativa.

4.4 Ameaças à Validade do Estudo

Nesta seção apresentamos as principais ameaças identificadas que poderiam comprometer a validade do estudo com base em (CHATZIGEORGIOU, 2019):

Validade de Construção: Entre as ameaças identificadas nesta categoria está a possibilidade de que as questões de pesquisa ou os termos e sinônimos da *string* de pesquisa sejam inadequados, amplos ou incompletos e o fato da avaliação dos estudos não ser conduzida em pares. Para mitigar essas ameaças, buscamos a orientação de pesquisadores experientes na área e utilizamos a ferramenta *QAnubis*, que permite realizar uma análise qualitativa de forma sistematizada, evitando abordagens *ad-hoc*.

Validade Interna: Para mitigar estas ameaças, aplicamos um formulário para extração de dados levando em consideração as QPs. Também aplicamos uma estratégia de busca de trabalhos relacionados às palavras-chave definidas na *string* e excluímos a

literatura cinza. Esse fator não teve impacto significativo, pois os estudos passaram por um rigoroso processo de seleção.

Validade Externa: A ameaça potencial está relacionada a verificar se os estudos selecionados de fato mencionam soluções de autenticação para arquiteturas de microsserviços. Para mitigar a generalização dos resultados, aplicamos as melhores recomendações propostas por (KITCHENHAM; BRERETON, 2013). Também foram aplicados critérios de qualidade para selecionar estudos que realmente atendessem às abordagens das questões de pesquisa mapeadas.

Validade de Conclusão: De acordo com as recomendações propostas por (WOHLIN et al., 2012), adotamos medidas para reduzir o preconceito do pesquisador, que pode influenciar de maneira significativa as conclusões e ser uma ameaça à validade dos resultados. Para lidar com essa questão, realizamos uma leitura atenta dos estudos e utilizamos a tabulação dos dados extraídos sob supervisão do orientador para mitigar o viés interpretativo e responder adequadamente às questões de pesquisa (QPs).

4.5 Discussão

Realizamos uma RSL para identificar soluções de autenticação e autorização em arquitetura de microsserviços, obtendo 19 estudos primários. Observamos uma carência significativa de estudos relacionados a soluções de autenticação e autorização em arquiteturas de microsserviços. O modelo de arquitetura de microsserviços divide uma aplicação em serviços pequenos, independentes e escaláveis. Portanto, os aspectos de segurança devem ser cuidadosamente considerados para cada serviço individual, pois a arquitetura de microsserviços aumenta a superfície de ataque e a falta de atenção pode tornar a aplicação vulnerável. Um API *gateway* recebe solicitações de clientes e gerencia políticas de autenticação de usuário e controle de acesso para um conjunto de microsserviços.

Os resultados deste estudo revelam que diversas tecnologias suportam a implementação de soluções e ferramentas (todas listadas na Tabela 4). Com base na análise realizada, os estudos revelam que a ferramenta Keycloak, os protocolos *OpenID Connect* e *OAuth*, e o padrão JWT estão entre os estudos selecionados sobre autenticação e autorização de sistemas para arquitetura de microsserviços (ver Tabela 4). Além disso, destaca-se os desafios de Segurança/Privacidade e Autenticação e Autorização/Controle de Acessos como os mecanismos mais mencionados nos estudos primários selecionados. Com base na análise realizada, os estudos revelam que a ferramenta Keycloak, as tecnologias OAuth e JWT e os desafios de Segurança/Privacidade e Autenticação e Autorização/Controle de Acessos são os mecanismos mais mencionados entre os estudos selecionados sobre autenticação e autorização de sistemas para arquitetura de microsserviços. Em suma, a principal contribuição desta RSL foi a identificação das ferramentas, tecnologias, protocolos e os desafios enfrentados na autenticação e autorização em arquiteturas de microsserviços.

Para trabalhos futuros, pretendemos expandir esta pesquisa utilizando os 19 estu-

dos selecionados como base para um protocolo de *snowballing* (WOHLIN et al., 2016), a fim de identificar outros estudos relacionados. Além disso, ampliar as descobertas com uma busca mais abrangente em fontes de literatura cinza.

Estas conclusões orientarão as decisões sobre a implementação de uma solução abrangente para autenticação e autorização em um ambiente universitário, onde existem atualmente múltiplas soluções independentes e desarticuladas de sistemas legados para autenticação e autorização. A partir dessas ferramentas, será realizada uma analise técnica para identificar a tecnologia que melhor supre a demanda.

4.6 Trabalhos Relacionados

Notamos que na literatura contém vários estudos secundários (revisões ou mapeamentos) que investigam sobre arquitetura de microsserviços. No entanto, observamos um número limitado de trabalhos que se aprofundam especificamente no domínio da autenticação e autorização no contexto da arquitetura de microsserviços.

Recentemente, Almeida e Canedo (2022) realizaram uma RSL selecionando 24 estudos publicados após o ano de 2010, para buscar identificar estudos que abordam autenticação e autorização em arquitetura de microsserviços, quais os mecanismos de segurança utilizados para lidar com os desafios e quais tecnologias de código aberto que implementam os mecanismos. Os autores relatam também uma grande carência quando o estudo é especifico para autenticação e autorização em arquiteturas de microsserviços, constataram também poucas indicações na literatura sobre soluções de código aberto.

Pereira-Vale et al. (2019) descrevem um projeto e os resultados de um mapeamento sistemático para identificar os mecanismos de segurança usados em sistemas baseados em microsserviços. Selecionaram 26 estudos primários de um total de 321 estudos encontrados entre o ano de 2015 e 2018. Eles observaram um incremento na quantidade de publicações em torno de 50% ao ano e os mecanismos de segurança mais relatados são Autorização, Autenticação e Credenciais.

Alshuqayran, Ali e Evans (2016) realizaram um estudo de mapeamento sistemático sobre os estilo de arquitetura para microsserviços que se concentra na identificação de desafios arquitetônicos relacionados a sistemas de microsserviços. A revisão foi conduzida da seguinte forma: primeiramente foi elaborada as questões de pesquisa. Na sequencia foi preparada a estratégia de busca, que restringiu a trabalhos publicados entre 2014 e 2016. Após a aplicação dos critérios de exclusão e inclusão, 33 estudos foram selecionados. Uma avaliação qualitativa foi realizada para criar um modelo de esboço para a qualidade do trabalho.

Ponce et al. (2022) apresentaram uma revisão multivocal da literatura que busca organizar conhecimento sobre segurança em aplicativos baseados em microsserviços. Analisaram 58 estudos primários, publicados de 2011 até o final de 2020. A pesquisa da literatura branca foi realizada combinando a *string* de pesquisa com o título e o resumo. A

pesquisa na literatura cinza foi realizada pelos motores de busca: Google, Bing e DuckDuckGo. O estudo identificou dez maus cheiros para segurança de microsserviços e uma taxonomia que permite ajudar a mitigar seus efeitos.

Hannousse e Yahiouche (2021) realizaram um mapeamento sistemático para categorizar e apresentar um guia sobre ameaças conhecidas em microsserviços e como elas podem ser detectadas, mitigadas ou evitadas. A busca resultou 1.067 estudos, dos quais 46 foram selecionados como estudos primários entre o ano de 2011 e 2019. Para evitar a perda de estudos relevantes, foi conduzindo procedimento de *snowballing* (WOHLIN et al., 2016) recursiva para trás e para frente.

Na Tabela 6 apresentamos uma breve discussão comparativa e as soluções identificadas nos estudos relacionados.

Tabela 6 – Resumo comparativo dos trabalhos relacionados.

Estudos	Período	#	Escopo	Soluções
Nosso Estudo	2015-2023	19	Identificar soluções, ferramentas e tecnologias que lidam com autenticação e autorização para arquitetura de microsserviços.	Keycloak, Zuul, Kong and Apache APISIX.
Alshuqayran (2016)	2014-2016	33	O estudo se concentra na identificação de desafios arquitetônicos relacionados a sistemas de microsserviços.	API Gateway, OAuth e Proxy.
Pereira-Vale (2019)	2015-2018	26	Descreve um projeto e seus resultados para identificar os mecanismos de segurança utilizados em sistemas baseados em microsserviços.	API Gateway, Distributed Session, SSO e JWT.
Hannousse (2021)	2011-2019	46	Fornece um guia sobre ameaças conhecidas em microsserviços e como elas podem ser detectadas, mitigadas ou evitadas.	API Gateway.
Ponce (2022)	2011-2020	58	Apresenta uma taxonomia referente à segurança em aplicações que utilizam a arquitetura de microsserviços.	API Gateway, OAuth e OpenID Connect.
de Almeida e Canedo (2022)	2010-2021	24	Aborda desafios, mecanismos e tecnologias relacionados à autenticação e autorização em microsserviços.	API Gateway, OAuth, OpenID Connect e JWT.

Tabela 7 – Objetivos por Estudos

Objetivos por Estudos	**Nosso Estudo (2023)**	**Alshuqayran (2016)**	**Pereira-Vale (2019)**	**Hannousse (2021)**	**Ponce (2022)**	**de Almeida e Canedo (2022)**
Ferramentas de gerenciamento de autenticação para microsserviços	●	○	○	○	○	◐
Tecnologias utilizadas para autenticação em microsserviços	●	○	●	○	○	●
Desafios relacionados a autenticação em microsserviços	●	●	○	●	●	●

Legenda: **Sim** = ●| **Parcial** = ◐| **Não** = ○

Por fim, a Tabela 7 sumariza e compara os objetivos abordados por cada estudo. Cada estudo foi classificado quanto aos objetivos propostos, com uma categorização em "Sim", "Parcial" ou "Não", de acordo com a aderência em que o estudo abrange o assunto. Os trabalhos relacionados foram identificados por meio de uma busca *ad-hoc*, utilizando a mesma *string* de buscar aplicada na seleção dos estudos primários da RSL.

5 DESENVOLVIMENTO DO TRABALHO

Este capítulo apresenta os detalhes da proposta sobre o desenvolvimento de uma *API Gateway* para os *webservices* do SIE. Conforme identificado nos desafios apontados em resposta a `QP3` na RSL, o `Controle de Acesso` é um dos aspectos abordados neste trabalho.

A introdução de uma camada de *software* para controle de acesso surge como uma necessidade da UFSM de modernizar a arquitetura dos *webservices* do SIE. A Divisão de Análise e Desenvolvimento de Sistemas do CPD, enfrenta um desafio crucial: a falta de uma abordagem que permita a troca de informações entre sistemas de forma segura. Questão essencial para a modernização dos sistemas. Para isso, é fundamental adotar um controle de acesso adequado para comunicação entre os sistemas institucionais.

O relatório do CISC abordado no Capítulo 1, trás as recomendações e sugestões de melhorias possíveis que podem ser aplicadas ao SIE *web* como forma de aprimorar os aspectos de segurança da informação.

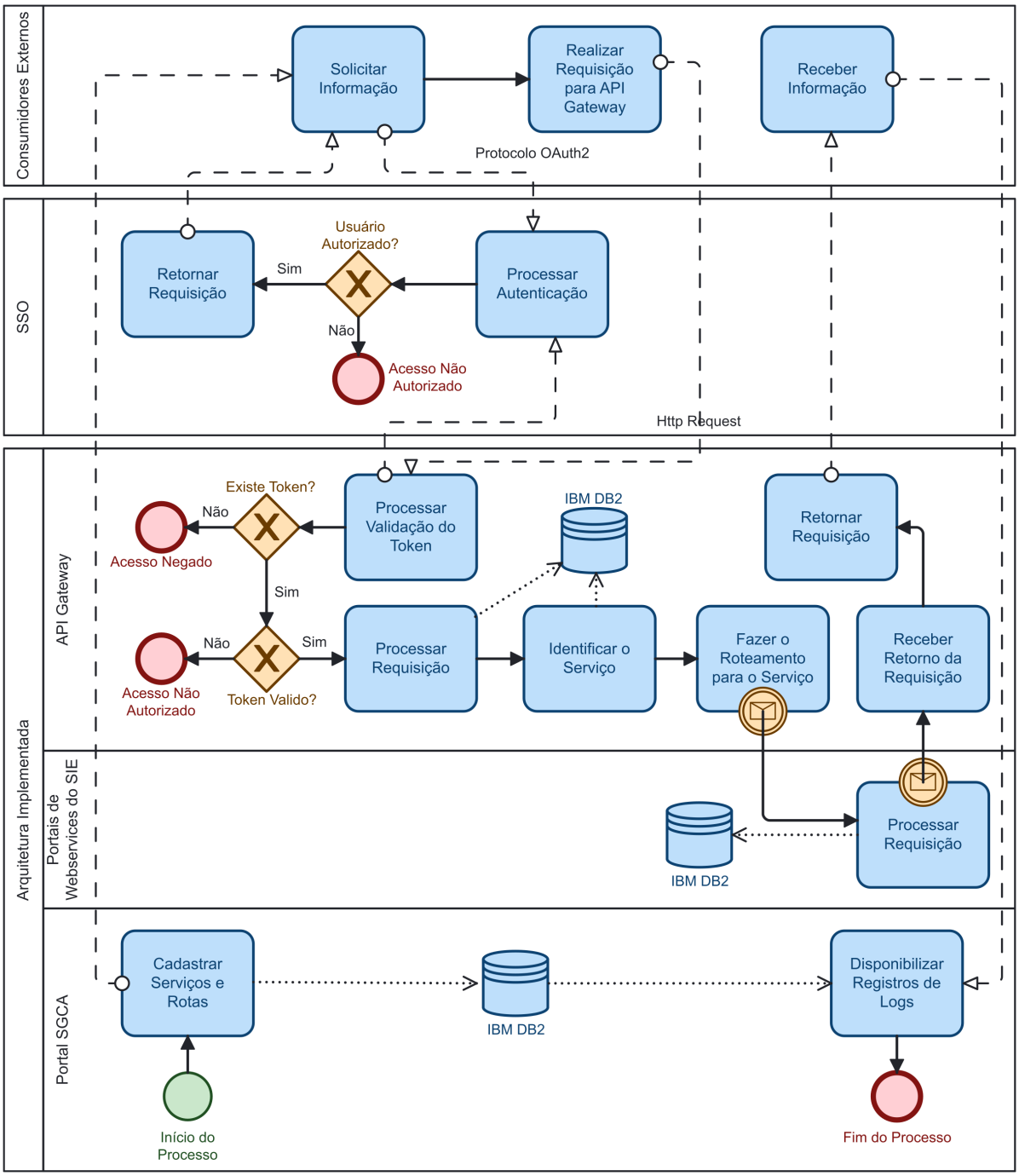
O SIE *web* é divido em portais e utiliza um modulo chamado SIE *core* que implementa todas as regras de negócio por meio de chamadas *Enterprise JavaBeans* (EJB), os portais *web* acessam diretamente o SIE *core*. Os acessos externos realizados aos *webservices* do SIE, como os do sítio institucional e o aplicativo *mobile*, precisam passar pelo SSO e a *API Gateway*.

O portal do SSO é um projeto que permitirá os usuários acessar vários aplicativos com um único *login*, simplificando o processo de autenticação e aprimorando o controle de acesso. Este serviço vai oferecer autenticação JWT, OAuth 2.0, interna para os portais do SIE *web* e GOV.br. A aplicação SSO está sendo concebida para provisionamento no *cluster kubernetes* do CPD, ela também oferece uma API REST para comunicação. Atualmente está em desenvolvimento, com previsão de conclusão para o segundo semestre deste ano e de implantação em produção com previsão para o primeiro semestre do ano de 2025.

O portal do Sistema Gerenciador de Controle de Acesso (SGCA) é um sistema que gerencia usuários, grupos de usuários, aplicações, autorizações e configurações de acesso aos dados do sistema de acordo com o perfil de cada usuário.

Para demonstrar a proposta de implementação foi elaborado um diagrama de processo utilizando a notação BPMN ilustrado na Figura 7, a qual descreve o processo com três participantes: Os `Consumidores Externos` representa os usuários que iniciam o processo solicitando informações. O `SSO` é responsável pela autenticação dos usuários, além de gerenciar todo o ciclo de vida do *token* JWT, como disponibilização, renovação e revogação. E a `Arquitetura Implementada` com as raias *API Gateway*, Portais de *webservices* do SIE e Portal SGCA. O diagrama detalha como as requisições são tratadas e autenticadas por meio dos fluxos, melhorando o controle de acesso e o encaminhamento correto da informação solicitada pelos consumidores externos.

Figura 7 – Diagrama de Processos BPMN da Arquitetura Implementada



Fonte: Autor

O processo começa com a atividade de cadastrar serviços e rotas no portal de SGCA. Quando um consumidor externo precisa de uma informação, primeiro ele passa pelo processo de autenticação por meio do SSO. Se o usuário for autorizado, a requisição é retornada e encaminhada para a *API Gateway*, caso contrário, finaliza o processo com uma mensagem de acesso não autorizado.

A *API Gateway* verifica se o *token* de autenticação está presente. Em caso afirmativo, prossegue para verificar se o *token* é válido por meio do SSO, mas se ele não existir

finaliza com uma mensagem de acesso negado. Se o *token* é válido contínua para processar a requisição, mas se o *token* não é válido encerra a requisição com uma mensagem de acesso não autorizado. Após a checagem da autenticação a *API Gateway* realiza o processamento da requisição, gera o log, identifica para qual serviço deve ser roteada por meio de consulta ao banco de dados e executa a requisição para o serviço apropriado.

O portal de *webservice* do SIE recebe a requisição, processa e retorna ela para a *API Gateway*, que recebe e encaminha a resposta ao consumidor externo. Dessa forma, a informação é disponibilizada para o consumidor, o registro de log da requisição fica disponível para consulta no portal de SGCA e o processo é finalizado.

Os fluxos de mensagens adicionais realizam a comunicação entre os serviços internos e a interação com a base de dados. O portal de SGCA é o responsável por realizar a gestão do cadastro de serviços e consultar o banco de dados para disponibilizar os registros de *log*. O banco de dados tem a responsabilidade de manter as informações armazenadas.

A principal contribuição proposta por este projeto é centralizar todas as solicitações de requisições externas com destino aos *webservices* do SIE em uma *API Gateway*. Com isso adicionar uma camada de controle de acesso para gerenciar o que está sendo consumido, evitar expor informações internas a clientes externos, reduzir os riscos contra acesso não autorizado e uso indevido de dados. O desenvolvimento deste projeto está com o cronograma alinhado com o do portal de SSO e conta com o apoio de um servidor e um bolsista da Divisão de Análise e Desenvolvimento de Sistemas do CPD da UFSM.

5.1 Requisitos de Software

Os Requisitos de *Software* (RS) foram definidos com base na literatura utilizada neste trabalho e no conhecimento prévio dos pesquisadores envolvidos na condução do estudo. Os requisitos estão diretamente associados às decisões de projeto.

- **RS1. A *API Gateway* deve atuar como um único ponto de entrada para solicitações de acesso aos portais do SIE *web*.** A proposta tem como foco fornecer um controle de acesso centralizado. Dessa forma todas as requisições devem ser registradas e passam por um filtro de validação de autenticação antes de serem encaminhada para o serviço de destino;
- **RS2. A *API Gateway* deve checar a solicitação para verificar se está autenticada.** Ponto crucial de uma *API Gateway*, pois é fundamental que apenas solicitações validadas sejam encaminhadas para o serviço desejado;
- **RS3. A *API Gateway* deve gerenciar os serviços e rotas.** Permitir realizar a gestão dos serviços e rotas (cadastrar, editar, listar, excluir e desabilitar);
- **RS4. A *API Gateway* deve realizar o roteamento das requisições para os devidos serviços.** A aplicação deve apresentar uma maneira de direcionar as requisições para o serviço apropriado;
- **RS5. A *API Gateway* deve registrar todas as requisições de aces-**

sos realizadas. O sistema deve armazenar as informações de cada requisição, gerando um *log* dos acessos para controle de acesso, rastreabilidade e auditoria.

5.2 Decisões de Projeto

Esta seção descreve as Decisões de Projeto (DP) para desenvolver um sistema de *API Gateway* que suporta todos os requisitos apresentados na Seção 5.1. Para cada decisão de projeto, indicamos os Requisitos de *Software* (RS) associados:

- **DP1.** Dentre as ferramentas identificadas na RSL, a que mais se aproximou das necessidades dos sistemas institucionais foi a *Kong API Gateway*. No entanto, optou-se por desenvolver um sistema próprio, considerando fazer parte de um projeto mais amplo e sua integração com outros portais do SIE *web* (**RS1**). Para desenvolver a *API Gateway* proposta, foi utilizado o *framework Spring Boot* com JDK 21. Essa decisão foi adotada em função da tecnologia ser utilizada pela equipe de desenvolvimento do CPD da UFSM e pela integração com as tecnologias utilizadas nos demais sistemas existentes na instituição;
- **DP2.** Toda solicitação que chega à *API Gateway* deve ser verificada sua autenticidade. Solicitações em que não é possível identificar um *token* válido devem receber um retorno de acesso proibido. Para implementar esta funcionalidade foram utilizadas as tecnologias de *GlobalFilter* e *GatewayFilter* disponibilizadas por meio do *framework Spring Boot* (**RS2**);
- **DP3.** A gestão dos serviços e rotas foi desenvolvida com o *framework Spring Boot* e JDK 11 (**RS3**), essa funcionalidade foi implementada junto ao portal do SGCA do SIE que utiliza estas tecnologias. Essa funcionalidade tem a capacidade de fornecer uma base de conhecimento dos serviços operacionalizados pela *API Gateway*;
- **DP4.** As funcionalidades de persistência de dados são realizadas por meio do Sistema Gerenciador de Banco de Dados (SGBD) DB2 na versão 11.5.90 da empresa IBM (**RS3 e RS5**). Para armazenar as informações das operações da *API Gateway* foi utilizado o banco de dados institucional, já existente, utilizado pelos demais sistemas desenvolvidos pelo CPD da UFSM;
- **DP5.** Para desenvolver o roteamento das requisições ao serviço adequado (**RS4**), foi utilizada a biblioteca *Spring Cloud Gateway*, disponibilizada peloo *framework Spring Boot* e executada na plataforma JDK 21;
- **DP6.** A *API Gateway* deve fornecer uma funcionalidade para registro de *logs* (**RS5**). O sistema deve armazenar as informações da requisição, como: quem deseja acessar, de qual endereço está acessando, qual o serviço deseja acessar e disponibilizar esses registros para fins de rastreabilidade e auditoria. Para implementar este mecanismo foi adotado o *framework Spring Boot* e JDK 21.

5.3 Arquitetura do Software

Diante da necessidade de um controle de acesso para os *webservices* do SIE e do contexto deste trabalho, desenvolvemos um diagrama na Figura 8, utilizando a ferramenta Structurizr[1], com a notação *C4 model* (VáZQUEZ-INGELMO; GARCíA-HOLGADO; GARCíA-PEñALVO, 2020). A Structurizr é uma *Domain Specific Languages* (DSL) que permite gerar visualizações gráficas para diferentes notações arquiteturais (FOWLER; PARSONS, 2011). Neste caso, a visualização foi renderizada com o PlantUML (ROQUES, 2024). Esse diagrama demonstra o funcionamento da arquitetura com a *API Gateway* desenvolvida e sua integração com os demais sistemas da Instituição. A seguir, descreveremos cada um dos elementos que a compõem:

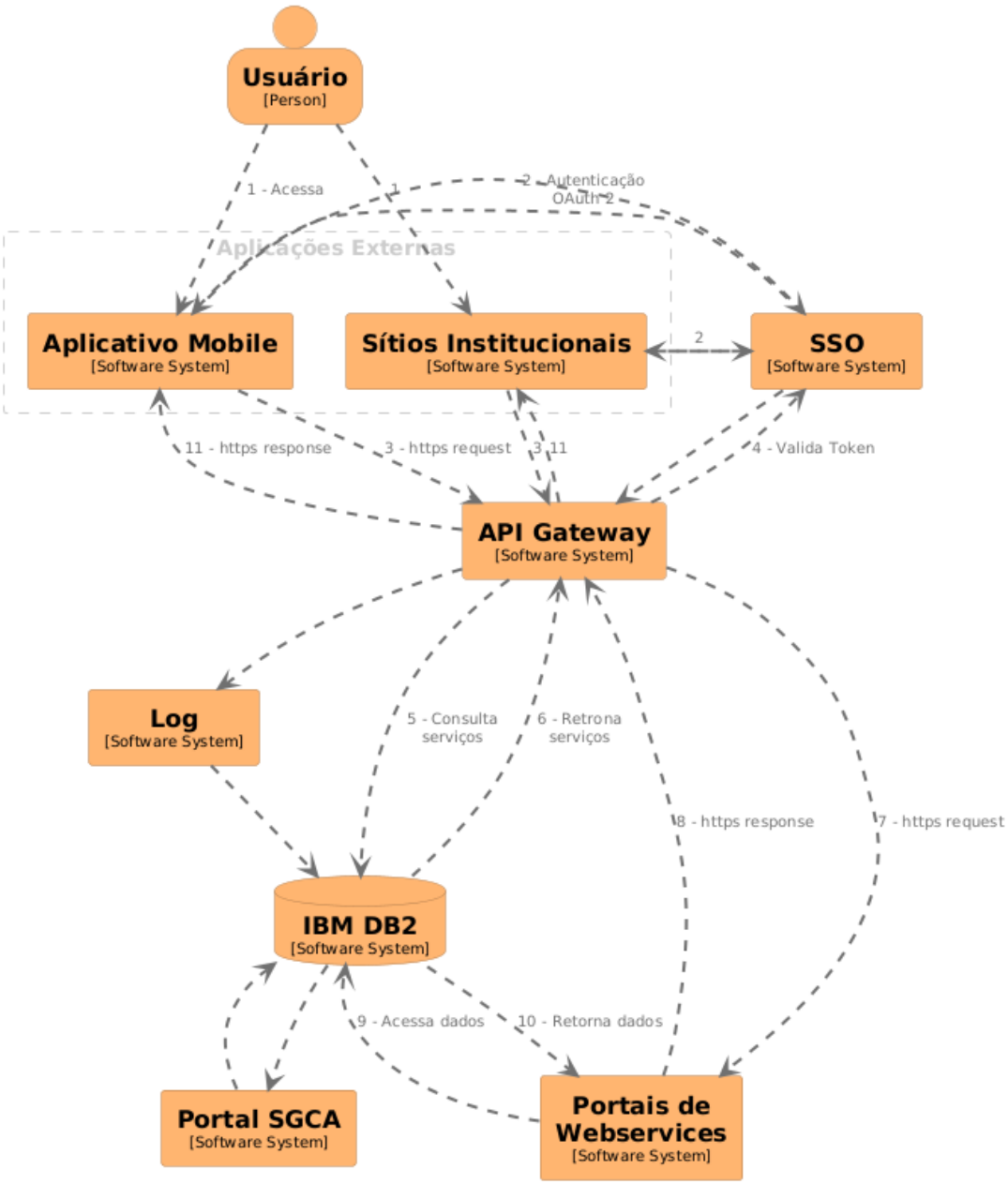
- `Usuário`: é o agente que interage com o aplicativo *mobile* e o sítio institucional;
- `Aplicações externas`: são os sistemas que necessitam se comunicar com os *webservices* do SIE para obter informações;
- `SSO`: é a aplicação que fornece um esquema de autenticação unificada;
- *`API Gateway`*: aplicação projetada para oferecer uma camada de controle de acesso por meio da validação de um *token* e armazenar todas as informações de cada solicitação, gerando *log* dos acessos;
- `IBM DB2`: é o SGBD responsável pela persistência de dados dos sistemas institucionais desenvolvidos pelo CPD da UFSM;
- `Portais` *`webservices`*: sistemas de *back-end* que possuem permissões de acesso para interagir com o banco de dados e são os responsáveis por receber as solicitações processá-las e retornar as informações;
- `Portal SGCA`: realiza a gestão do cadastro de serviços que é utilizado pela *API Gateway*. Também disponibiliza uma tela com a funcionalidade para acessar os registros de *log*;
- `Log`: tem a função de coletar e persistir em banco de dados as informações dos acessos.

O aplicativo *mobile* e os sítios institucionais da UFSM são as aplicações externas que irão consumir os portais de *webservices* do SIE via *API Gateway*. A *API Gateway* tem o papel de filtrar o tráfego, garantindo que apenas solicitações autorizadas cheguem aos serviços de *back-end*.

Quando o usuário acessa o aplicativo *mobile* ou um site institucional (1), ele é direcionado de forma transparente para a tela de *login* do portal de SSO por meio do protocolo OAuth 2.0 para se autenticar (2). Após o processo de autenticação bem sucedido, o SSO retorna para a aplicação um cabeçalho com o campo *authorization*. Este campo *authorization* contém um *token* JWT, que é emitido pelo SSO e representa a identidade do usuário autenticado.

[1] Structurizr: <https://structurizr.com/>

Figura 8 – Arquitetura Implementada



Fonte: Autor

É importante destacar que o SSO é responsável tanto pela emissão quanto pela validação desse *token* JWT. Quando uma aplicação externa realiza uma requisição via *API Gateway* (3), o *token* JWT presente no cabeçalho da requisição é validado pela *API Gateway*, que consulta o SSO para verificar se o *token* é válido e não expirado (4). Se o *token* for inválido ou estiver expirado, a *API Gateway* rejeita a solicitação e retorna uma resposta de acesso negado (11). Caso o *token* seja validado com sucesso pelo SSO, a *API Gateway* permite que a solicitação seja encaminhada para o serviço de *back-end* correspondente (7).

A *API Gateway*, ao receber a solicitação, realiza uma consulta ao banco de dados

(5) para identificar o serviço apropriado (6) e direciona a requisição para o serviço correto. Esse serviço de *back-end*, um dos portais de *webservices* do SIE, acessa o banco de dados (9) para obter as informações necessárias. O banco de dados retorna as informações solicitadas (10) ao portal de *webservice*, que então processa a solicitação e envia a resposta final ao cliente (11) por meio da *API Gateway* (8).

5.4 Implementação

A *API Gateway* está sendo projetada em uma arquitetura que permite seu provisionamento no *cluster kubernetes* do CPD, funcionando como um ponto de entrada centralizado para mitigar os riscos associados ao controle de acesso aos *webservices* do SIE. Assim, o *cluster kubernetes* garantirá a disponibilidade e a escalabilidade tanto para o *API Gateway* quanto para as demais aplicações do SIE.

Para desenvolver a *API Gateway* proposta foi escolhida a tecnologia do *Spring Cloud Gateway*, um projeto que tem como base o *Framework Spring Boot*. O *Spring Cloud Gateway* fornece bibliotecas para construir uma *API Gateway* no modelo de programação reativa usando o *Spring WebFlux*, que permite realizar muitas conexões simultâneas com eficiência. O objetivo do *Spring Cloud Gateway* é fornecer uma maneira simples e eficaz de roteamento para serviços, além de tratar questões sobre segurança e monitoramento.

O *Spring Cloud Gateway* é o sucessor da *API Gateway Zuul* dentro do ecossistema *Spring Boot*, que foi a segunda ferramenta mais mencionada na RSL realizada neste estudo. Diante dos resultados obtidos na RSL, optou-se por utilizar o *Spring Cloud Gateway* por ser o sucessor da *Zuul* e por sua compatibilidade com o desenvolvimento do SIE. O *Spring Cloud Gateway* oferece desempenho e suporte a recursos modernos, como roteamento dinâmico e sem bloqueio, que são fundamentais para atender às demandas atuais de sistemas distribuídos (SANCHEZ; CARNELL, 2021). Além disso, a escolha do *Spring Boot* é estratégica, pois facilita a integração com o projeto SSO e com outros sistemas da Instituição que foram desenvolvidos com essa tecnologia. Essa compatibilidade simplifica futuras manutenções e adaptações, com menor custo e esforço, além de ser uma solução familiar para a equipe de desenvolvimento.

Para armazenar as configurações de serviços e rotas da *API Gateway* foi modelada a tabela `API_GATEWAY` no banco de dados institucional `DBSM` conforme Figura 9. Para operacionalizar essa funcionalidade, foi utilizado o SGBD DB2 na versão 11.5.90.

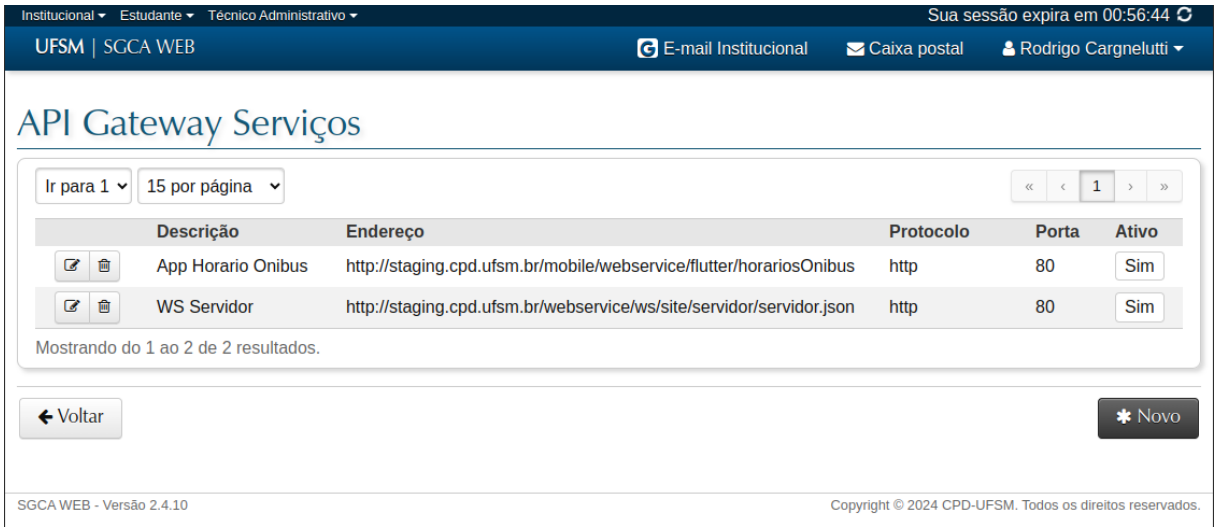
A gestão de configuração (adicionar, listar, atualizar e desativar), serviços e rotas, ilustrada nas Figuras 10 e 11 e utilizada pela *API Gateway*, foi implementada no portal SGCA do SIE. Estas informações são persistidas no banco de dados da instituição conforme estrutura ilustrada na Figura 9. Nas linhas 1 e 2 do código fonte ilustrado na Figura 12, o método `myRoutes` recebe dois parâmetros: `builder` e `apiGatewayService` e retorna um objeto `RouteLocator`. `RouteLocator` é uma classe utilizada para localizar e definir rotas. `RouteLocatorBuilder` é uma classe do *Spring Cloud Gateway* utilizada para

Figura 9 – *Script SQL* da tabela *API_GATEWAY*

```
1  CREATE TABLE DBSM.API_GATEWAY (
2      ID_API_GATEWAY   INTEGER NOT NULL CONSTRAINT PK_API_GATEWAY PRIMARY
           KEY,
3      SV_DESCRICAO     VARCHAR(100) NOT NULL,
4      SV_PROTOCOLO     VARCHAR(25) NOT NULL,
5      SV_HOST          VARCHAR(50) NOT NULL,
6      SV_PATH          VARCHAR(100) NOT NULL,
7      SV_PORTA         VARCHAR(10) NOT NULL,
8      SV_ATIVO         CHAR(1) NOT NULL DEFAULT 'S',
9      RT_DESCRICAO     VARCHAR(100) NOT NULL,
10     RT_PROTOCOLO     VARCHAR(25) NOT NULL,
11     RT_PATH          VARCHAR(100) NOT NULL,
12     RT_METODO        VARCHAR(40) DEFAULT NULL,
13     DH_CRIACAO       TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
14     DT_ALTERACAO     DATE NOT NULL,
15     HR_ALTERACAO     TIME NOT NULL,
16 );
```

Fonte: Autor

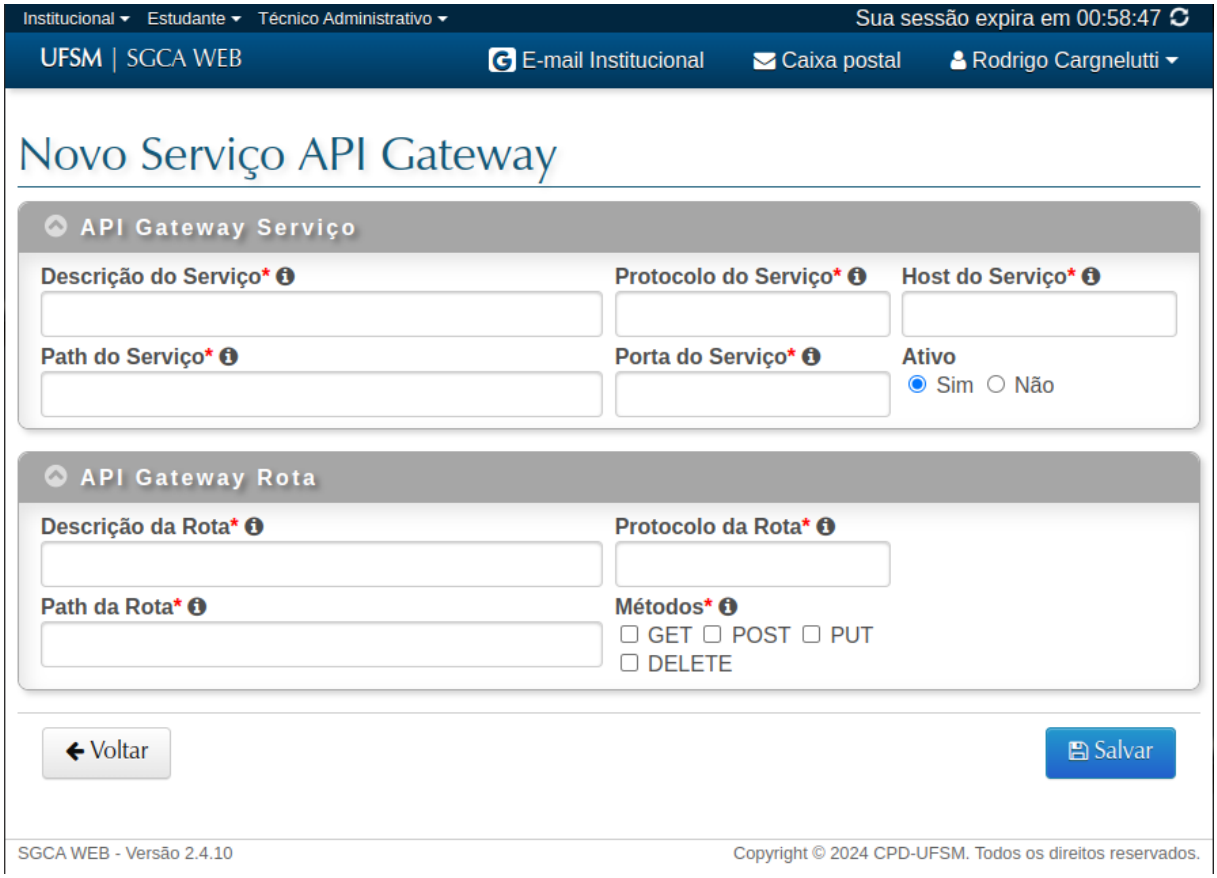
Figura 10 – Portal SGCA - Lista dos Serviços



Fonte: Autor

construção de rotas. Geralmente é usada em conjunto com a `RouteLocator` para definir a configuração de roteamento. A classe `ApiGatewayService` é usada para obter a lista de serviços e rotas configuradas. Este método configura dinamicamente as rotas usando informações obtidas de um serviço. Na linha 3 do código Java, o método `listAll` é invocado para obter uma lista de serviços. Na linha 4, `builder.routes` retorna um objeto do tipo `RouteLocatorBuilder.Builder`, uma classe interna de `RouteLocatorBuilder` no *Spring Cloud Gateway*, usada para configurar múltiplas rotas. Na linha 5 temos a iteração na lista de serviços para buscar cada serviço. Na expressão *lambda* `p -> p` da

Figura 11 – Portal SGCA - Formulário de Cadastro



Fonte: Autor

Figura 12 – Método que implementa o roteamento

```
1  public RouteLocator myRoutes(RouteLocatorBuilder builder,
2  ApiGatewayService apiGatewayService) throws JsonProcessingException {
3      var listaServicos = apiGatewayService.listAll();
4      var routes = builder.routes();
5      for (var servico : listaServicos) {
6          routes.route(p -> p
7              .path(servico.getRT_PATH())
8              .filters(f -> f.setPath(servico.getSV_PATH()))
9              .uri(servico.getSV_URI()));
10     }
11     return routes.build();
12 }
```

Fonte: Autor

linha 6, o `p` representa um objeto usado para configurar condições específicas para uma rota, como o caminho que ela deve seguir. Os métodos encadeados das linhas 7 a 9 não podem ser chamados em qualquer ordem. Cada método cumpre uma função específica que exige uma sequência lógica para configurar a rota corretamente. A linha 7 define

o caminho da rota que a *API Gateway* deve corresponder. A linha 8 aplica um filtro, adicionando o caminho da rota para o valor especificado e na linha 9 é definido o endereço de destino para onde a solicitação deve ser encaminhada. Em suma, para cada serviço na lista, é criada uma rota, aplicado um filtros e definido o endereço de destino. Na linha 11 é realizado o retorno do objeto. O tratamento de exceções deve ser realizado por meio da aplicação que chama a *API Gateway*.

Para realizar a verificação do *access token* da solicitação, foi utilizada a tecnologia de filtros do *framework Spring Boot*. No código fonte ilustrado na Figura 13 foi implementado um filtro para verificar se o campo *Authorization* está presente no cabeçalho da solicitação e realizar a validação do *token*. Dessa forma, utilizando a tecnologia de filtros, a *API Gateway* desenvolvida oferece uma camada adicional de segurança por meio do controle de acesso, pois consegue filtrar o tráfego antes que ele alcance o serviço de *back-end*.

Figura 13 – Classe que implementa a verificação do *token* de *Authorization*

```
1  public class CheckAuthorizationFilter implements GlobalFilter, Ordered {
2
3      public int getOrder() { return -1; }
4
5      public Mono<Void> filter(ServerWebExchange exchange,
           GatewayFilterChain chain) {
6          if (!exchange.getRequest().getHeaders().containsKey("
               Authorization")) {
7              throw new ApiGatewayForbiddenException();
8          }
9          String authorizationHeader = exchange.getRequest().getHeaders().
               get("Authorization").getFirst();
10         String bearerToken = AuthorizationBearerUtils.
               extractBearerTokenFromAuthorization(authorizationHeader);
11         if (!isValidToken(bearerToken)) {
12             throw new ApiGatewayUnauthorizedException();
13         }
14         return chain.filter(exchange);
15     }
16 }
```

Fonte: Autor

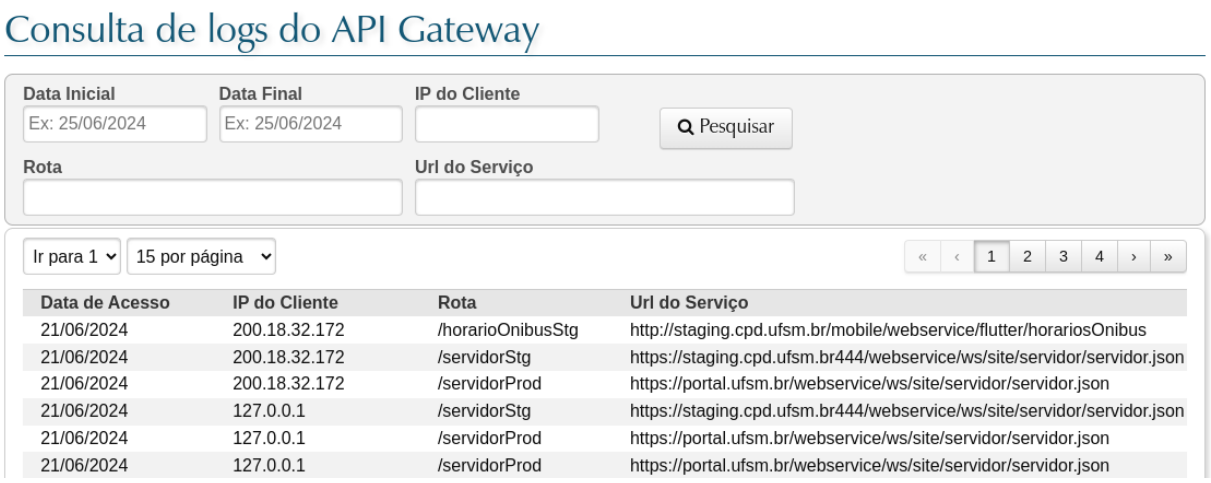
A classe `CheckAuthorizationFilter` da linha 1 da Figura 13 implementa duas interfaces, a `GlobalFilter` e a `Ordered`. `GlobalFilter` é um filtro global que é aplicado a todas as solicitações que chegam a *API Gateway*. `Ordered` especifica a ordem de execução do filtro. Na linha 3, o método `getOrder` é utilizado para determinar a prioridade de execução do filtro. Neste caso foi definido como `-1` indicando que o filtro tem alta prioridade, e assegurando que esse filtro será um dos primeiros a serem executados na cadeia de filtros.

Na linha 5 foi implementado o método `filter`, para realizar a verificação e a validação do campo *Authorization* da solicitação. O método `filter` retorna `Mono<Void>`, indicando que ele executa operações de forma assíncrona, permitindo que o método seja não bloqueante e dessa forma outras operações podem continuar enquanto o filtro é processado. Na linha 6, é verificado se o campo *Authorization* está presente na solicitação, se o campo não estiver presente, é lançada uma exceção de acesso proibido. Se a solicitação contém o campo *Authorization* então é extraído o *token Bearer*.

Na linha 11, o método `isValidToken` é invocado para verificar se o *token* é válido. Se o *token* não for válido, linha 12, é lançado uma exceção de resposta informando que não tem autorização. Se o *token* for válido, linha 14, retorna `chain.filter(exchange)` sinalizando para prosseguir com o filtro.

Para rastrear e monitorar as solicitações que chegam a *API Gateway*, foi implementada uma funcionalidade para coletar e registrar as informações dos acessos. Esses registros são armazenados na tabela `API_GATEWAY_LOGS` do banco de dados institucional `DBSM` e as informações disponibilizadas em uma interface desenvolvida no portal SGCA conforme ilustra a Figura 14, acessível por meio do menu ***`API Gateway`*** `-> Consulta Logs`.

Figura 14 – Portal SGCA - Consulta de *Logs*



Fonte: Autor

A linha 1 do código fonte ilustrado na Figura 15, a classe pública `LogAcessoFilter` implementa duas interfaces, a *GlobalFilter* e a *Ordered*, que já foram explicadas no relato da classe `CheckAuthorizationFilter` ilustrada na Figura 13. Nas linhas 3 e 4 temos os atributos privados, `listaServicos` para armazenar a lista de serviços e `logAcessoService` para armazenar informações do *log* gerado e na linha 6 o método construtor da classe.

Na linha 13 temos o método `filter`, que é utilizado para registrar os *logs* de acesso. Na linha 14 é capturado e armazenado na variável `ipCliente` o endereço *Internet Protocol* (IP) do cliente requisitante. Na linha 17 é obtida a rota e na linha 18 é obtido o

Figura 15 – Classe que implementa a geração de *log*.

```
1  public class LogAcessoFilter implements GlobalFilter, Ordered {
2
3      private final List<ApiGatewayModel> listaServicos;
4      private final LogAcessoService logAcessoService;
5
6      public LogAcessoFilter(ApiGatewayService apiGatewayService,
           LogAcessoService logAcessoService) {
7          listaServicos = apiGatewayService.listAll();
8          this.logAcessoService = logAcessoService;
9      }
10
11     public int getOrder() { return -1; }
12
13     public Mono<Void> filter(ServerWebExchange exchange,
           GatewayFilterChain chain) {
14         String ipCliente = exchange.getRequest().getRemoteAddress().
               getAddress().getHostAddress();
15         LogAcessoModel logAcesso = new LogAcessoModel();
16         logAcesso.setIpCliente(ipCliente);
17         Route rota = exchange.getAttribute("org.springframework.cloud.
               gateway.support.ServerWebExchangeUtils.gatewayRoute");
18         Long id = (Long) rota.getMetadata().get("id");
19         ApiGatewayModel servico = buscaServicoById(id);
20         logAcesso.setServico(servico);
21         logAcessoService.geraLogAcesso(logAcesso);
22         return chain.filter(exchange);
23     }
24 }
```

Fonte: Autor

identificador da rota. Na linha 19 o método `buscaServicoById` é invocado para buscar o serviço correspondente a identificação da rota e de posse das informações necessárias, na linha 21 é gerado o *log* de acesso.

O uso dos *logs* gerados por meio da *API Gateway* pode contribuir para o monitoramento, rastreabilidade, auditoria, e outras atividades. A geração de *logs* é uma prática essencial para qualquer tipo de sistema, pois esses registros fornecem informações sobre os acessos realizados, permitindo análises para aprimorar o desempenho e a gestão dos serviços. Os *logs* gerados pela *API Gateway* possibilitam o monitoramento das requisições realizadas, permitindo rastrear cada solicitação, identificar sua origem, destino e quaisquer erros ocorridos. Isso é fundamental para: (i) identificar quais serviços são mais consumidos, por quem, em quais períodos e com que frequência; (ii) identificar comportamentos anômalos, como um aumento repentino no número de requisições, o qual poderia indicar problemas de desempenho ou um possível ataque.

Os *logs* são fundamentais para auditoria, assegurando que todas as transações e

operações sejam registradas de forma adequada. Os *logs* também são uma ferramenta que ajuda a entender o desempenho dos serviços. Com base nas informações de utilização registradas é possível prever a necessidade de escalabilidade dos serviços para atender a demanda. Em suma, os *logs* gerados pela *API Gateway* permitem uma gestão da infraestrutura, possibilitando auditorias e aprimoramento dos serviços.

Dessa forma, a implementação da *API Gateway* com as tecnologias e metodologias descritas proporciona o gerenciamento das rotas e serviços, geração de *logs* e maior controle de acesso aos *webservices* do ecossistema do SIE *Web*. Com isso, assegura-se que os requisitos e os objetivos definidos foram plenamente atendidos e também proporciona-se uma base para futuras expansões e melhorias.

5.5 Resumo do Capítulo

Este capítulo apresenta o desenvolvimento de uma *API Gateway* para o SIE *Web*, destacando a importância do controle de acesso na comunicação entre os sistemas da UFSM. É Abordado sobre os desafios enfrentados pela Divisão de Análise e Desenvolvimento de Sistemas do CPD. Ressalta-se a necessidade de uma abordagem de controle de acesso para os *webservices* do SIE.

Os requisitos de software foram elencados e detalhados, com base na RSL e no conhecimento dos pesquisadores envolvidos. Os principais requisitos incluem a verificação do *token* de autenticação, gerenciamento de serviços e rotas, e registro das requisições para auditoria. O projeto inclui uma camada de controle de acesso para comunicação entre os sistemas institucionais e a importância da integração da *API Gateway* com o portal de SSO para validar o *token* de autenticação.

A implementação utiliza o *framework Spring Boot* e JDK 21, com armazenamento de dados no SGBD IBM DB2. A proposta envolve a validação de *token* de acesso na *API Gateway*, assegurando que apenas solicitações autenticadas sejam processadas. A arquitetura melhora o controle de acesso e a rastreabilidade das requisições, centralizando as solicitações externas e gerenciando acessos de forma eficiente.

6 AVALIAÇÃO DE DESEMPENHO

Este capítulo apresenta a avaliação de desempenho da *API Gateway* desenvolvida em comparação com a *Kong API Gateway*. A proposta é analisar métricas de desempenho, como tempos de resposta, taxa de requisições por segundo e consumo de recursos computacionais, sob diferentes cargas de trabalho. O processo segue os princípios de engenharia de software experimental do Wholin (WOHLIN et al., 2012), com a definição de questões de pesquisa, coleta de dados e análise sistemática.

Objetivo: Avaliar o desempenho de duas ferramentas de *API Gateway* sob diferentes condições de carga para determinar qual oferece melhor desempenho em termos de latência, *throughput* e consumo de recursos.

Questões de Pesquisa (QP): Definimos as seguintes questões de pesquisa conforme apresentado na Figura 16:

QP1. Qual ferramenta apresenta menor tempo médio de resposta (*response time*) para as carga de trabalho de 100, 300 e 500 usuários simultâneos?
QP2. Qual ferramenta apresenta menor requisições por segundo (RPS) para as carga de trabalho de 100, 300 e 500 usuários simultâneos?
QP3. Qual ferramenta apresenta menor falhas por segundo (*failures*/s) para as carga de trabalho de 100, 300 e 500 usuários simultâneos?
QP4. Há diferenças significativas no consumo de recursos computacionais (memória, processamento e tráfego de rede) entre as soluções testadas?

Figura 16 – Questões de Pesquisa.

Tipo de Experimento: Experimentos *in virtuo* (TRAVASSOS; BARROS, 2003) envolvem a interação entre participantes e ambientes virtuais que simulam a realidade por meio de simulações, softwares ou sistemas computacionais. No caso do experimento realizado, o foco foi na avaliação de ferramentas de software em um ambiente controlado, utilizando cargas de trabalho simuladas para medir desempenho, falhas e uso de recursos computacionais das duas ferramentas de *API Gateway* sob diferentes condições de carga.

6.1 Contexto Experimental

O experimento foi conduzido em um ambiente controlado, configurado para simular três cenários com cargas de trabalho (100, 300 e 500 usuários), em que foram testados dois *endpoints*.

6.1.1 Cenários

Para as avaliações, os sistemas foram configurados em uma máquina virtual instanciada na nuvem privada do CPD com a seguinte configuração, 40 GB de memória RAM, 12 núcleos de processamento, 150 GB de armazenamento em disco e o sistema operacional

Linux Debian 11 (Bullseye). O SIE, a *API Gateway* desenvolvida e a *Kong API Gateway* são executados em contêineres utilizando a versão 26.1.4 do *Docker*. Vale destacar que os sistemas são executados de forma isolada em seus respectivos contêineres, não ocorrendo competição por recursos computacionais.

O escopo foi definido em três cenários distintos `C1`, `C2` e `C3` para avaliar a eficácia do uso de uma *API Gateway* na arquitetura proposta para os *webservices* da UFSM.

C1: Enviar as requisições diretamente para os *webservices* do SIE, sem utilizar uma *API Gateway*;

C2: Efetuar as mesmas requisições utilizando a *API Gateway* desenvolvida neste trabalho, conforme descrito na Seção 5.4;

C3: Realizar as mesmas requisições por meio da ferramenta *Kong API Gateway* (ver Seção 4.3.1), identificada na RSL como a terceira mais mencionada e a que mais se assemelha ao objetivo e ao estilo arquitetônico deste trabalho.

Vale ressaltar que as *API Gateways*, a desenvolvida (C2) e a *Kong* (C3) foram configuradas com os mesmos serviços. Para coletar métricas de tempo de resposta, utilizamos a versão 2.25.0 do *Locust*, simulando a execução de várias requisições simultâneas. Para as avaliações, criamos *Tasks* e definimos o comportamento dos usuários que farão as requisições para cada cenário e *endpoint* por meio de *scripts* em *Python*, conforme ilustrado na Figura 17.

Da primeira a quarta linha da Figura 17 é realizado a importação das classes e funções essenciais. A linha 6 configura o intervalo de tempo de 5 segundos para gravar das informações em arquivos do tipo CSV. A linha 8 define a classe "`MyTask`", que herda de "`TaskSet`" e agrupa as tarefas que serão executadas pelo *Locust*. O decorador "`task`", da linha 9, sinaliza que o método é uma tarefa que será executada pelo *Locust* e a linha 10 define o método "`get_horarios_onibus`" que realiza requisição por meio do método "`GET`". O método "`GET`" é um método "`HTTP`" utilizado para solicitar dados de um determinado serviço. Da linha 11 a 16 é definido o cabeçalho "`HTTP`" que será enviado na requisição "`GET`". "`Content-Type`" especifica que os dados são do tipo "*application/json*", "`X-UFSM-Access-Name`" e "`X-UFSM-Device-ID`" são campos personalizados, usados para identificar o nome e o ID do dispositivo e o "`Authorization`" contém um token Bearer de autenticação. Na linha 17 é realizada a requisição "`GET`" para o *endpoint* `"/horarioOnibus"` com o cabeçalho "`headers`". Na linha 19 é definido a classe "`MyHttpUser`", que herda de "`HttpUser`". A linha 20 associa o conjunto de tarefas "`MyTask`" ao usuário "`MyHttpUser`" que executará as tarefas definidas em "`MyTask`". Na linha 21 definimos um intervalo de tempo entre 1 e 2 segundos de espera antes do usuário executar a próxima tarefa.

A linha 23 garante que o bloco de código só será executada quando o *script* for instanciado diretamente e não importado por outro *script*. Na linha 24 temos "`os.system`" que executa um comando no terminal, "`locust -f`" especifica o *script Locust* a ser exe-

Figura 17 – *Script Python* utilizado para avaliação por meio da ferramenta *Locust*.

```python
from locust import HttpUser, TaskSet, task, between
from datetime import datetime
import locust.stats
import os

locust.stats.CSV_STATS_INTERVAL_SEC = 5

class MyTask(TaskSet):
    @task
    def get_horarios_onibus(self):
        headers = {
            "Content-Type": "application/json",
            "X-UFSM-Access-Name": "abcabcabc",
            "X-UFSM-Device-ID": "01010101",
            "Authorization": "Bearer
               cTZ3NWU0cjY1cWU0NnJxNnc1ZTRyNjVxZTQ2cnE2dzVlNHI2NXFlND=="
        }
        self.client.get("/horarioOnibus", headers=headers)

class MyHttpUser(HttpUser):
    tasks = [MyTask]
    wait_time = between(1, 2)

if __name__ == "__main__":
    os.system(f"locust -f cenario2_mobile.py --host=http://staging.cpd.
       ufsm.br:9999 --headless -u 500 -r 4.17 -t 60m --html C2-U500-R4
       .17-T60-mobile_charts.html --csv C2-U500-R4.17-T60-mobile")
```

Fonte: Autor

cutado, "`host`" define a URL base do sistema que será testado, "`headless`" executa o *Locust* sem a interface *web*, "*u*" especifica que serão simulados 500 usuários simultâneos, "`-r`" define a taxa de geração de 4.17 usuários por segundo, "`t`" define o tempo total de execução do teste será de 60 minutos, "`html`" gera um relatório no formato `HTML` com gráficos dos resultados, "`csv`" gera arquivos no formato `CSV` com as métricas. Para iniciar o teste definido no *script*, é preciso executá-lo diretamente no terminal utilizando o comando "`python3 cenario2_mobile.py`", tendo como premissa que o arquivo com o *script* foi salvo com o nome "`cenario2_mobile.py`" e no mínimo a versão 3.8 do *Python* instalada.

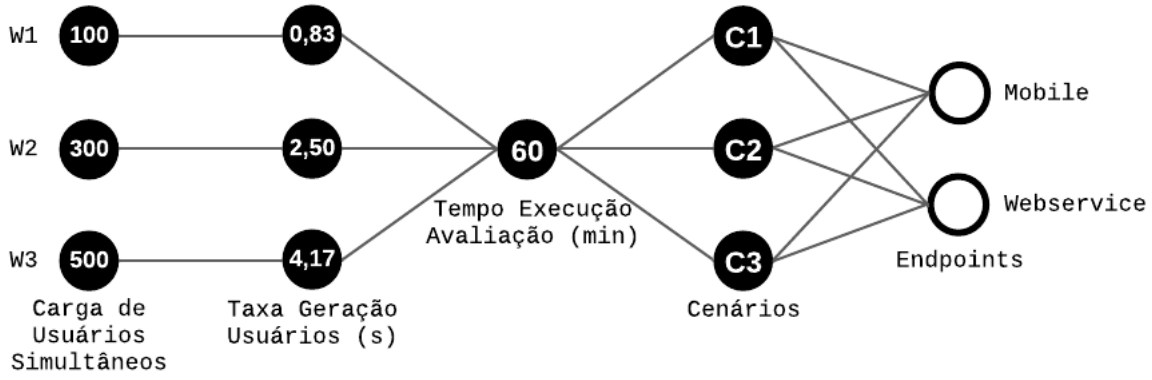
6.1.2 Cargas de Trabalho (Workloads)

As avaliações foram realizadas de forma segmentada por Cargas de Trabalho (*workloads*): 100 (W1), 300 (W2) e 500 (W3) usuários simultâneos, cenários (C1, C2 e C3) e *endpoints* (`"/horarioOnibus"` - portal *Mobile* e `"/servidor"` - portal *webservice*). Os *workloads* escolhidos refletem cenários reais de uso dos sistemas da UFSM. Determina-

mos que todos os usuários serão gerados em um período de 120 segundos (2 minutos). A taxa de geração de usuários por segundos (*Ramp Up Time*) foi calculada dividindo a quantidade total de usuários pelo tempo para que todos os usuários entrem em operação simultaneamente. Todos os testes foram executados por um período de 60 minutos (1 hora) para assegurar a estabilidade nos resultados.

A primeira execução W1 foi realizado com uma carga de trabalho (*workload*) de 100 usuários simultâneos, uma taxa de geração de 0,83 usuários por segundo (*Ramp Up Users*), abrangendo os três cenários e os dois serviços. Por sua vez a segunda execução W2 foi conduzida com uma carga de 300 usuários simultâneos, gerados a uma taxa de 2,50 usuários por segundo, cobrindo os três cenários e os dois *endpoints*. Por fim, a terceira execução W3 foi executada com uma carga de 500 usuários simultâneos, gerados a uma taxa de 4,17 usuários por segundo, abrangendo os três cenários e os dois *endpoints*. O diagrama da Figura 18 ajuda a compreender como as avaliações foram realizadas com as cargas de trabalho (*workloads*) de 100, 300 e 500 usuários simultâneos. Já a Tabela 8 apresenta um panorama de todas as avaliações realizadas, agrupadas por *workloads*, *endpoints* e cenários.

Figura 18 – Cargas de trabalho (*Workloads*) das avaliações



Fonte: Autor

Coletamos as métricas sobre o `Número Total de Requisições` *(Requests)* , `Número Total de Falhas` *(Fails)* , `Tempo Médio de Resposta` *(Average)* , `Taxa de Requisições por Segundo` *(Current RPS)* e a `Taxa de Falhas por Segundo` *(Current Failures/s)* .

As figuras que apresentam as avaliações de cada cenário são compostas por dois gráficos, o primeiro apresenta as métricas do `Total de Requisições por Segundo`, com duas linhas, a linha na cor verde é sobre a métrica de `Requisição por Segundo (RPS)`, já a linha na cor vermelha é sobre as `Falhas por Segundo (`*`Failures/s`*`)`. Por sua vez, o segundo gráfico é referente as métricas de `Tempo de Resposta`, a linha na cor lilas é referente a métrica *`95th Percentile`* e a linha na cor amarela diz respeito ao `Tempo Médio de Resposta` (*`Average Response Time`*). Complementarmente, os resultados serão

Tabela 8 – Avaliações agrupadas por *Workloads*, *Endpoints* e Cenários

Avaliação	Workload	Usuários Simultâneos	Endpoints	Cenários
A01	W1	100	Mobile	C1
A02				C2
A03				C3
A04			Webservice	C1
A05				C2
A06				C3
A07	W2	300	Mobile	C1
A08				C2
A09				C3
A10			Webservice	C1
A11				C2
A12				C3
A13	W3	500	Mobile	C1
A14				C2
A15				C3
A16			Webservice	C1
A17				C2
A18				C3

apresentados em uma tabela e três gráficos, um para cada carga de trabalho (*workload*), associados a seus respectivos cenários e *endpoint*.

6.1.3 Serviços em Teste (*Service Under Test - SUT)*

Nesta seção abordaremos o contexto dos serviços submetidos aos testes de desempenho. Os *webservices* em questão desempenham um papel importante no ambiente institucional, fornecendo funcionalidades essenciais para diferentes aplicações utilizadas por um vasto público. A análise de desempenho desses serviços é motivada pela necessidade de garantir a eficiência dos *webservices* que suportam as operações da Instituição. Esses serviços são responsáveis por fornecer informações e funcionalidades a diversas aplicações, servindo tanto à comunidade acadêmica quanto a administrativa.

Um dos serviços testados é o `Servidor`[1] do portal de *webservice*. Este serviço recebe uma solicitação contendo o parâmetro `contrato` com o número de contrato e, em resposta, fornece informações detalhadas sobre um servidor específico. O serviço desempenha um papel essencial, pois é amplamente utilizado por diversas aplicações que necessitam de informações sobre os servidores da instituição. Essas aplicações incluem, mas não se limitam a: Sistema de Recursos Humanos, Sistema de Pagamento, Sistema Acadêmico e Portal de Transparência.

O público desse serviço inclui tanto servidores administrativos e docentes, quanto usuários externos, como estudantes e cidadãos interessados em informações públicas. A confiabilidade e a rapidez desse serviço são cruciais para o bom funcionamento das operações diárias da Instituição.

[1] Servidor: <https://portal.ufsm.br/webservice/ws/site/servidor/servidor.json>

O segundo serviço submetido aos testes de desempenho é o de `Horários de Ônibus`[2] que fornece informações sobre os horários de ônibus, disponibilizado por meio do portal *mobile*. O principal consumidor deste serviço é o aplicativo *mobile* da UFSM, amplamente utilizado por estudantes, docentes e servidores administrativos. O serviço de horários de ônibus é uma funcionalidade essencial dentro do ecossistema de aplicações móveis da UFSM. Dado o crescente uso de dispositivos móveis, é essencial que o serviço de consulta aos horários de transporte público que atendem à universidade funcione de maneira eficiente, fornecendo informações em tempo real, que são fundamentais para o planejamento diário dos usuários.

A análise de desempenho deste serviço visa garantir sua operação de forma eficiente, de forma a atender às necessidades de um grupo de usuários cada vez mais dependente de informações móveis em tempo real. Os serviços em teste são componentes vitais do ecossistema digital da UFSM, e a análise de seu desempenho garante que eles continuem atendendo às necessidades da Instituição de forma eficiente.

6.1.4 Ferramenta Locust

Para coletar métricas na avaliação de desempenho, usamos a versão 2.25.0 da ferramenta *Locust*[3], utilizada para testes de desempenho em aplicações *web* (MEIER, 2007). Projetada para simular um grande número de usuários acessando um serviço simultaneamente, ela permite avaliar como a aplicação se comporta sob alta demanda. Além disso, oferece a criação de cenários realistas com métricas em tempo real, os quais podem ser programados por meio de *scripts* em *Python*.

Durante a execução do teste, o *Locust* coleta e exibe métricas de desempenho, como tempo de resposta, taxa de requisição por segundo, taxa de sucesso/falha, percentis e erros de requisição. Essas métricas podem ser visualizadas em tempo real na interface web do *Locust* e exportadas em formatos CSV e HTML para análise. Após o teste, é possível analisar os dados coletados para identificar gargalos, falhas de desempenho e funcionalidades do sistema em teste que precisam de otimização. Vale ressaltar que escolhemos a ferramenta *Locust* para a análise de desempenho devido à sua flexibilidade, simplicidade e por ser de código aberto.

Locust é utilizado para atestar que aplicações possam suportar o tráfego esperado e identificar possíveis problemas antes que eles afetem os usuários finais. Ela permite que os cenários de teste sejam facilmente personalizados, oferecendo uma abordagem intuitiva para a simulação de múltiplos usuários concorrentes. Além disso, conta com uma interface web amigável para monitoramento em tempo real, facilitando a análise contínua dos testes.

[2] Horários de Ônibus: <https://portal.ufsm.br/mobile/webservice/flutter/horariosOnibus>
[3] Locust: <https://locust.io>

6.2 Avaliações

Com base nas QPs, coletamos as seguintes métricas para análise:

(i) **QP1.** `Tempo Médio de Resposta` *(Average Response Time)*,

(ii) **QP2.** `Taxa de Requisições por Segundo` *(Current RPS)* e

(iii) **QP3.** `Taxa de Falhas por Segundo` *(Current Failures/s)*.

Os resultados são apresentados em tabelas e gráficos, organizados por carga de trabalho e tipo de *endpoint*.

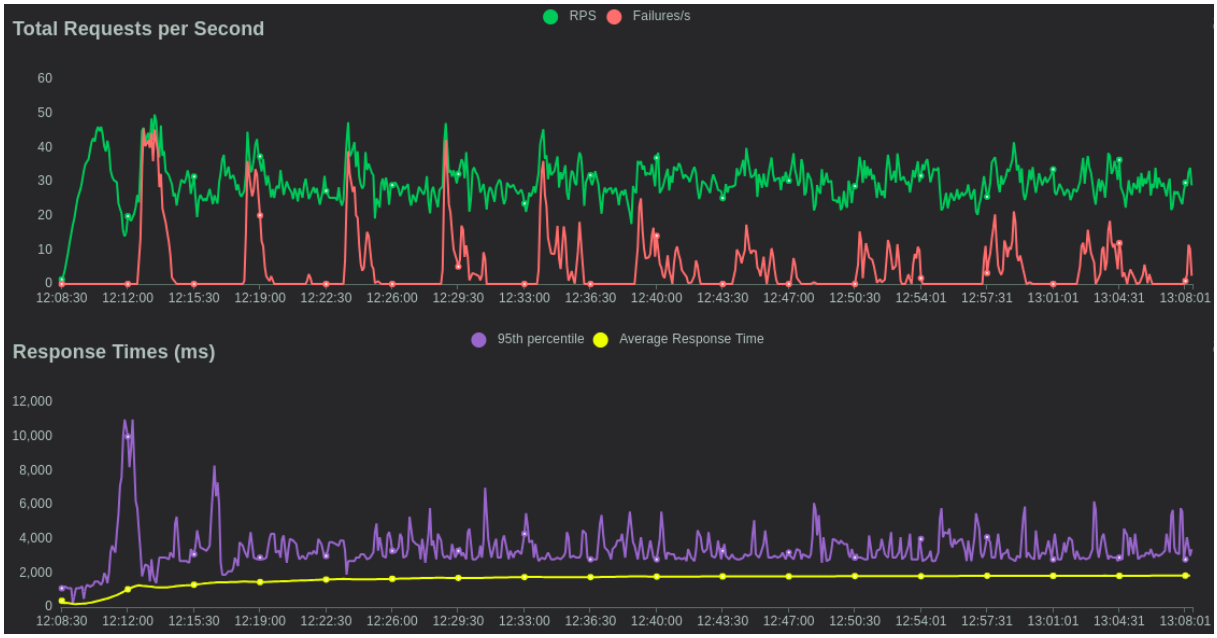
6.2.1 Avaliações com 100 Usuários Simultâneos - *Endpoint Mobile*

A análise das métricas de desempenho coletadas para a carga de trabalho (*Workload*) de 100 usuários simultâneos, considerando os três cenários e o *endpoint mobile* que busca informações dos horários de ônibus para o aplicativo UFSM Digital. Os resultados são apresentados na Tabela 9 e incluem o número total de requisições, número de falhas, tempo médio de resposta por segundo, taxa de requisições por segundo e taxa de falhas por segundo. As Figuras 19, 20 e 21 ilustram essas métricas graficamente.

Tabela 9 – Avaliações W1 100 Usuários Simultâneos - *Endpoint Mobile*

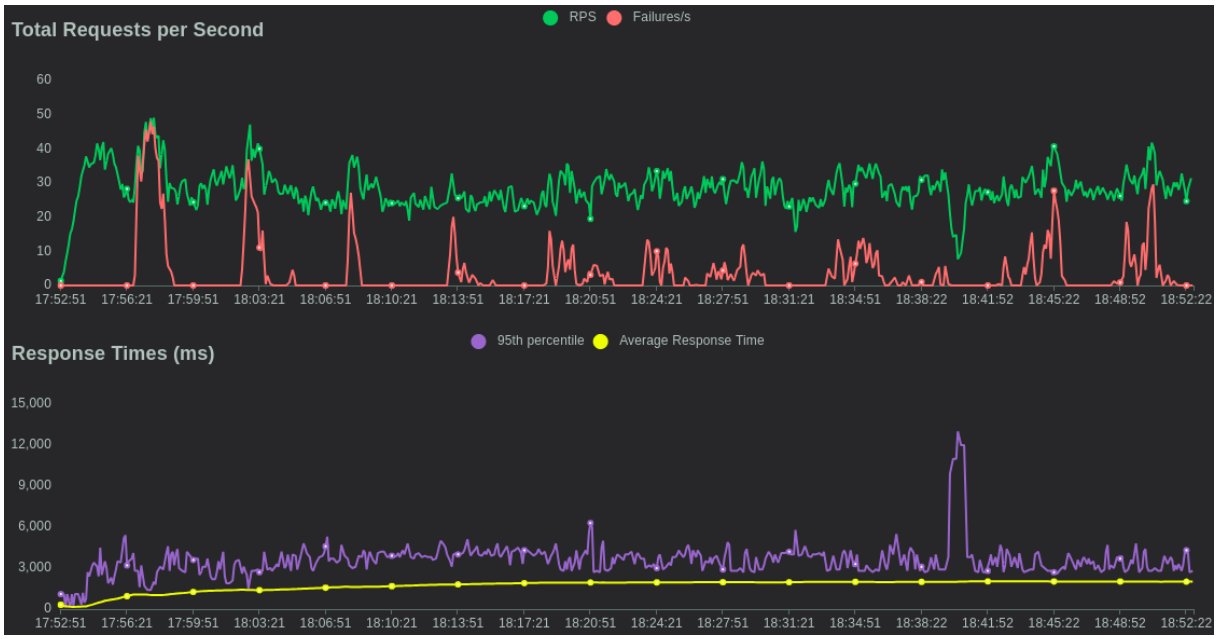
Avaliação	Cenário	Requisições	Falhas	Média(s)	Requisições(s)	Falhas(s)
A01	**C1**	105.510	15.721	1,85	29,31	4,37
A02	**C2**	100.859	12.867	2,00	28,02	3,57
A03	**C3**	104.658	15.909	1,88	29,07	4,42

Figura 19 – Avaliação A01: W1 100 Usuários - C1 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundos



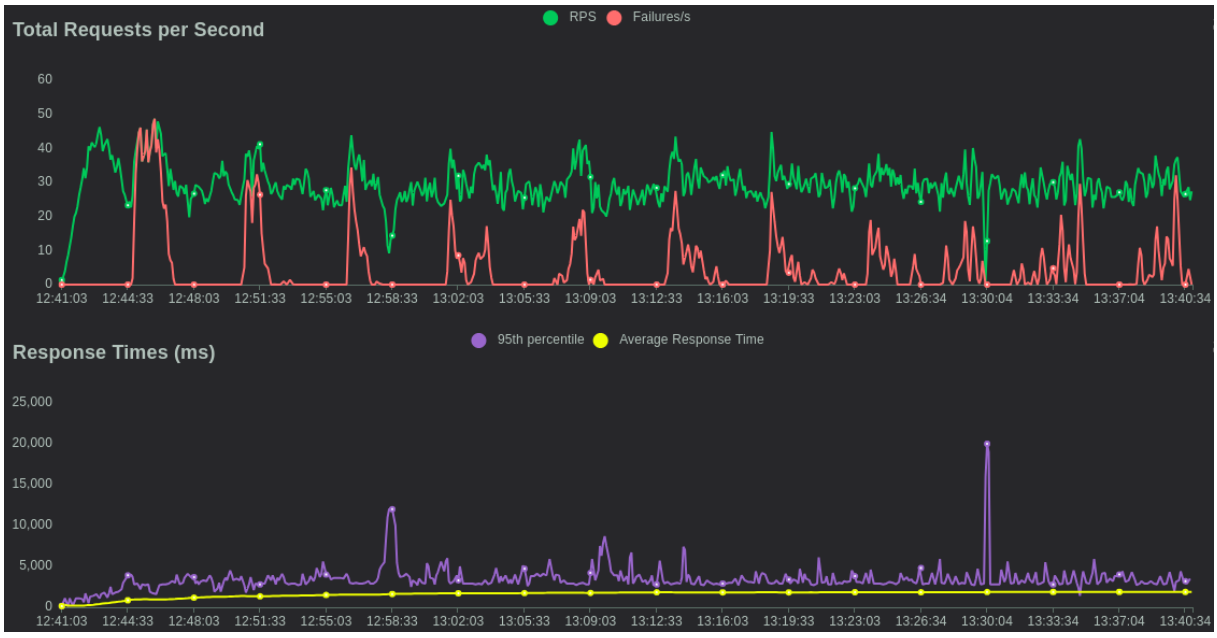
Fonte: Autor

Figura 20 – Avaliação A02: W1 100 Usuários - C2 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundos



Fonte: Autor

Figura 21 – Avaliação A03: W1 100 Usuários - C3 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundos



Fonte: Autor

No cenário em que as requisições são feitas diretamente ao *webservice Mobile*, sem a intermediação de uma *API Gateway* (C1), o sistema processou o maior número de requisições 105.510 com a taxa de requisições por segundo mais alta 29,31. No entanto, esse desempenho veio acompanhada de 15.721 falhas, resultando em uma taxa de 4,37 falhas por segundo. Apesar do tempo médio de resposta ser o mais rápido, 1,85 segundos,

a alta taxa de falhas compromete a confiabilidade do sistema.

A utilização da *API Gateway* desenvolvida (C2) resultou em uma redução no número de requisições processadas 100.859 e na taxa de requisições 28,02 por segundo, em comparação ao cenário sem *API Gateway* (C1). Essa redução pode ser atribuída ao *overhead* introduzido pela *API Gateway*. Houve também uma queda na taxa de falhas 3,57 por segundo e no número total de falhas 12.867. Embora o tempo médio de resposta tenha aumentado para 2,00 segundos, esse cenário demonstrou um bom equilíbrio entre desempenho e confiabilidade.

A *Kong* (C3) apresentou uma taxa de 29,07 requisições por segundo, muito próxima do cenário sem *API Gateway* (C1), apresentando uma boa capacidade de processamento. O tempo médio de resposta foi de 1,88 segundos, intermediário entre os cenários testados. No entanto, o número de falhas foi de 15.909 o maior entre os três cenários, resultando na maior taxa de falhas por segundo 4,42.

A análise dos gráficos dos três cenários (Figuras 19, 20 e 21) para o *endpoint mobile* revela um comportamento geral semelhante em relação as taxas de RPS. O cenário sem *API Gateway* (C1) apresenta o maior desempenho em termos de RPS, mas com maior instabilidade e variação nos tempos de resposta. Por outro lado, os cenários com *API Gateway* (C2 e C3) apresentam uma leve redução no desempenho, porém com maior estabilidade e confiabilidade, demonstrando menos variações tanto em RPS quanto em tempos de resposta.

6.2.2 Avaliações com 100 Usuários Simultâneos - *Endpoint Webservice*

As métricas resultantes da avaliação com a carga de trabalho de 100 usuários simultâneos para os três cenários e o *endpoint* `servidor` do portal de *Webservice*, que busca as informações de um determinado servidor, são exibidas na Tabela 10. Os gráficos ilustrando essas métricas estão apresentados nas Figuras 22, 23 e 24.

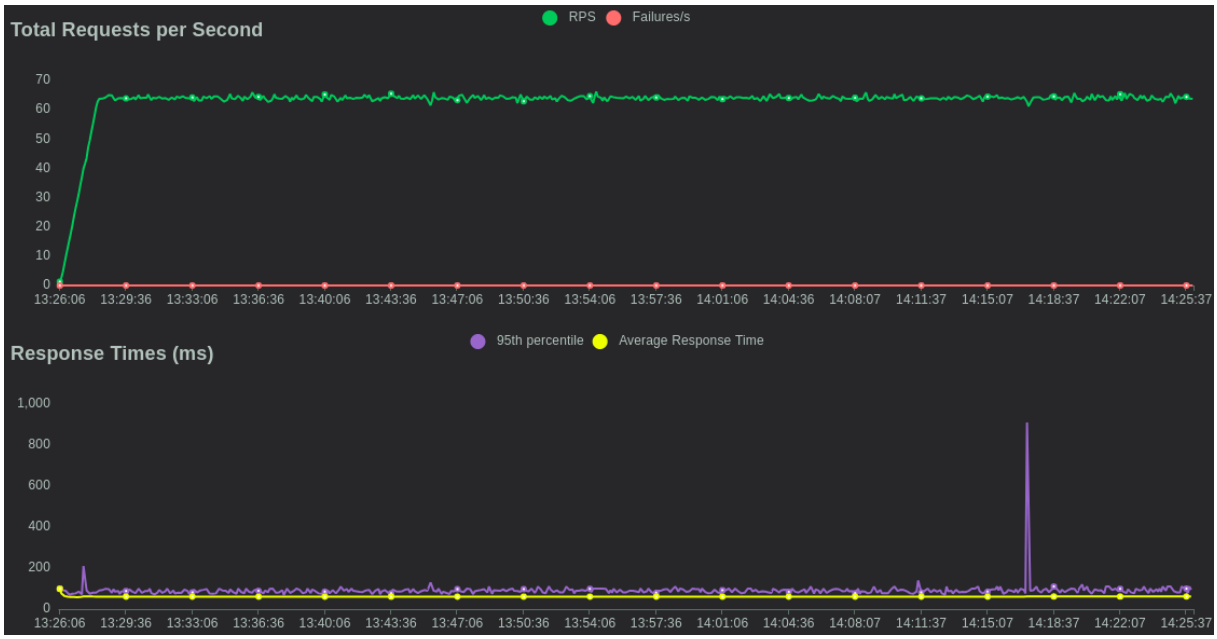
Tabela 10 – Avaliações W1 100 Usuários Simultâneos - *Endpoint Webservice*

Avaliação	Cenário	Requisições	Falhas	Média(s)	Requisições(s)	Falhas(s)
A04	**C1**	226.576	0	0,06	62,94	0
A05	**C2**	224.785	0	0,07	62,44	0
A06	**C3**	225.160	0	0,07	62,54	0

No cenário sem a intermediação de uma *API Gateway* (C1) apresentou o maior número de requisições 226.576 e a maior taxa de requisições por segundo 62,94. Além disso, não foram registradas falhas, o que indica eficiência e confiabilidade do serviço quando acessado diretamente. O tempo médio de resposta foi de 0,06 segundos, é o mais rápido entre os cenários, indicando que a ausência de intermediários reduz a latência.

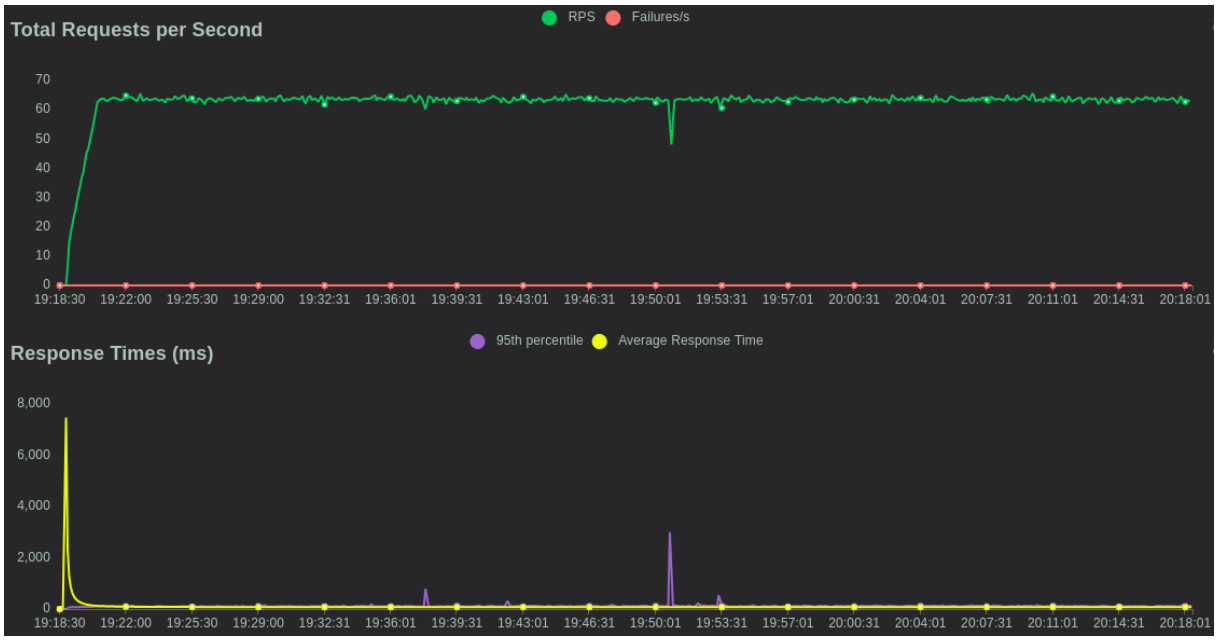
A implementação da *API Gateway* desenvolvida (C2) registrou um número ligeiramente menor de requisições 224.785 e uma taxa de 62,44 RPS um pouco mais baixa em

Figura 22 – Avaliação A04: W1 100 Usuários - C1 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

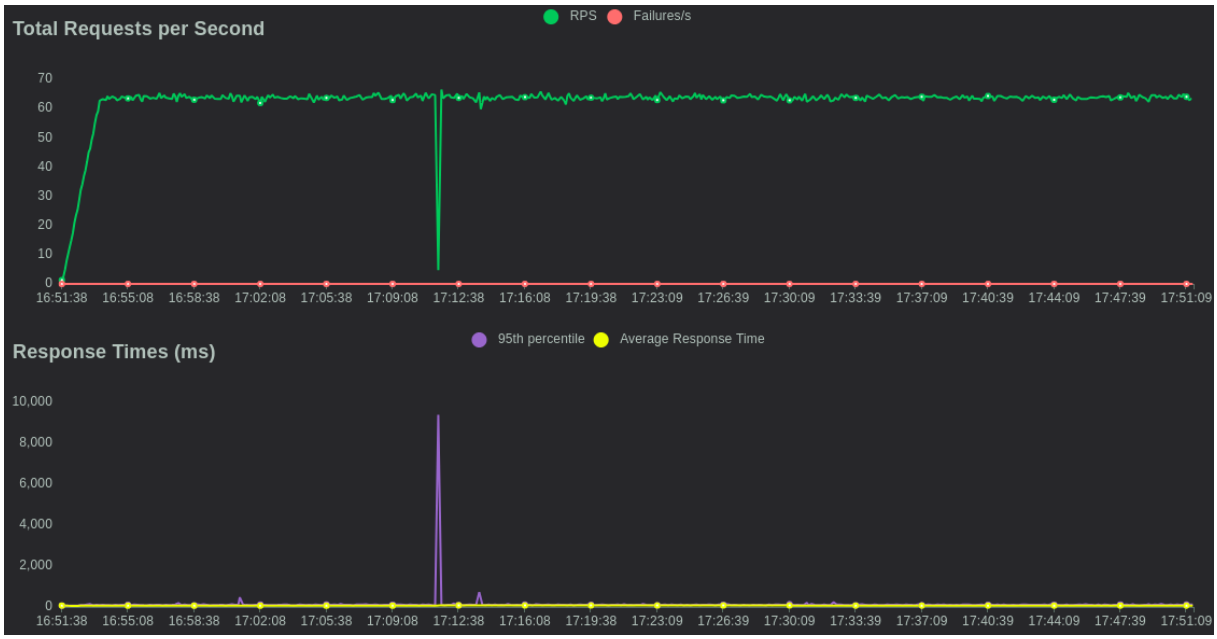
Figura 23 – Avaliação A05: W1 100 Usuários - C2 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

comparação ao cenário sem *API Gateway* (C1). No entanto, o tempo médio de resposta aumentou levemente para 0,07 segundos, mas ainda se manteve dentro do aceitável. Além disso, não foram registradas falhas, indicando que a *API Gateway* introduz um pequeno *overhead*, mas mantém a confiabilidade.

Figura 24 – Avaliação A06: W1 100 Usuários - C3 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

O uso da Kong (C3) resultou em um desempenho semelhante ao da *API Gateway* desenvolvida (C2), com um total de 225.160 requisições e uma taxa de 62,54 RPS, inferior ao cenário sem *API Gateway* (C1), mas superior à *API Gateway* desenvolvida (C2). O tempo médio de resposta foi de 0,07 segundos, alinhado com o desempenho da *API Gateway* desenvolvida (C2), e sem registro de falhas. Isso demonstra que a *Kong* é uma solução eficiente, oferecendo desempenho similar ao da *API Gateway* desenvolvida (C2). A Tabela 10 trás uma informação importante, a ausência de registros de falhas para os três cenários.

A análise dos gráficos (Figuras 22, 23 e 24) para o *endpoint Webservice* revela que todos os cenários apresentam um desempenho muito semelhante, tanto em termos de RPS quanto de tempos de resposta. A diferença mais notável é um leve aumento no tempo de resposta nos cenários com *API Gateway* (C2 e C3), mas sem comprometer a estabilidade e a confiabilidade.

6.2.3 Avaliações com 300 Usuários Simultâneos - *Endpoint Mobile*

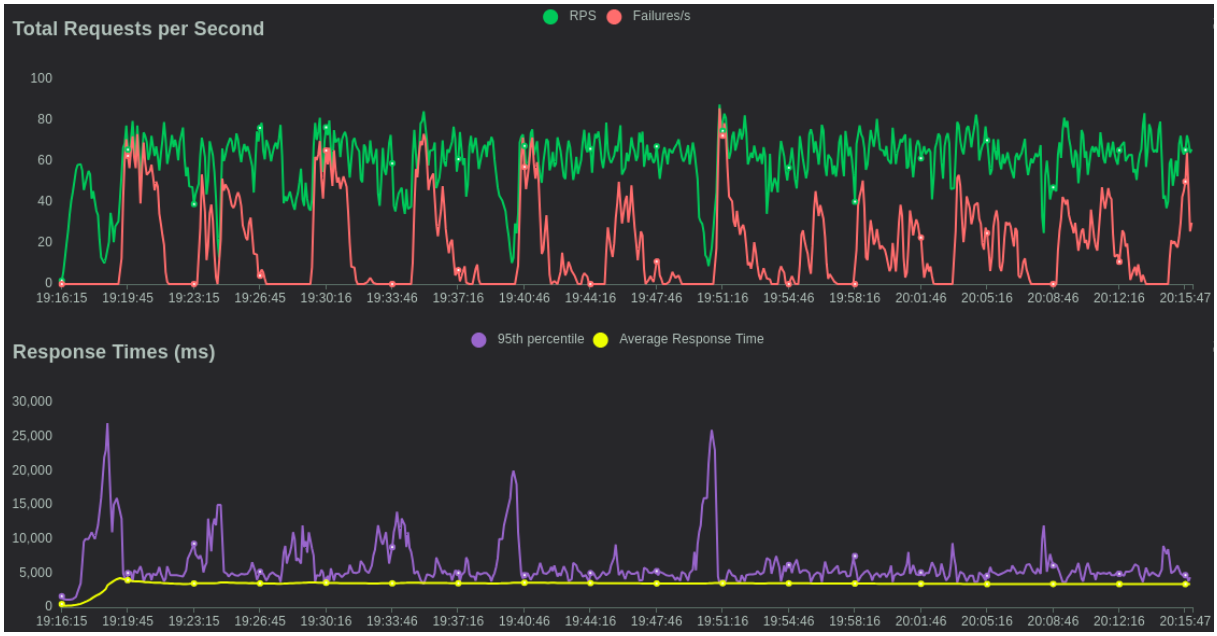
Analisamos as métricas de desempenho obtidas para a carga de trabalho de 300 usuários simultâneos para os três cenários e o *endpoint mobile* que busca os horários de ônibus para o aplicativo UFSM Digital. Os resultados são exibidos na Tabela 11 e os gráficos ilustrando essas métricas são apresentados nas Figuras 25, 26 e 27.

No cenário, onde as requisições são enviadas diretamente ao serviço (C1), o serviço processou 217.217 requisições, com uma taxa de 60,34 requisições por segundo. O tempo

Tabela 11 – Avaliações W2 300 Usuários Simultâneos - *Endpoint Mobile*

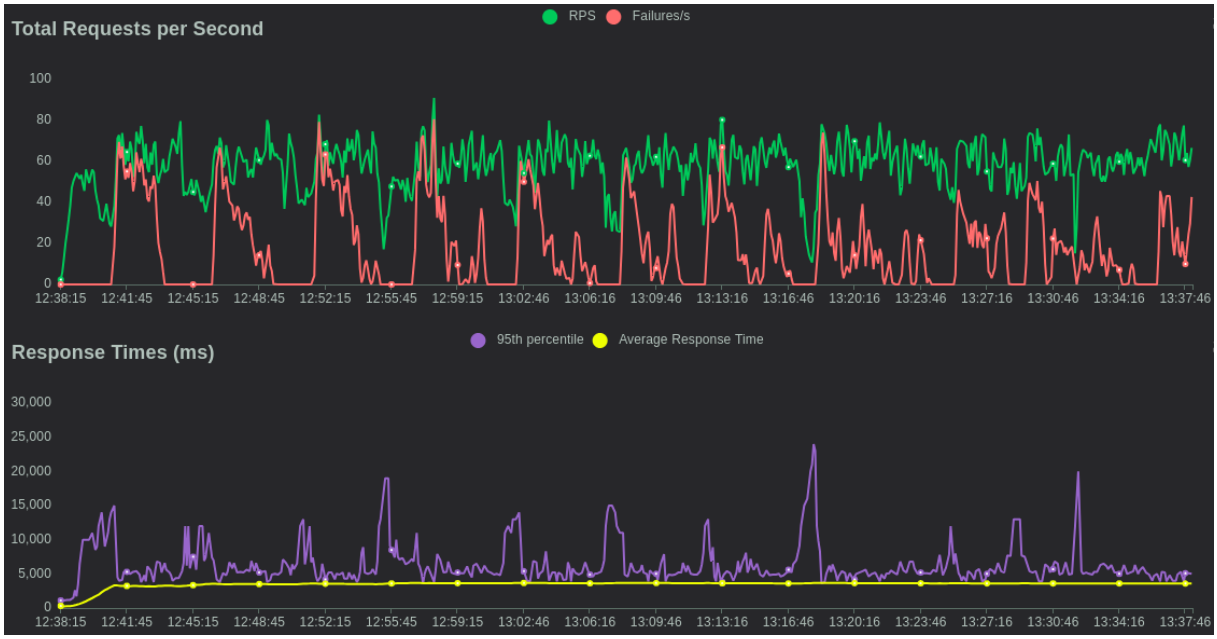
Avaliação	Cenário	Requisições	Falhas	Média(s)	Requisições(s)	Falhas(s)
A07	**C1**	217.217	62.743	3,38	60,34	17,43
A08	**C2**	208.769	61.556	3,58	57,99	17,10
A09	**C3**	215.038	64.346	3,25	59,73	17,87

Figura 25 – Avaliação A07: W2 300 Usuários - C1 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



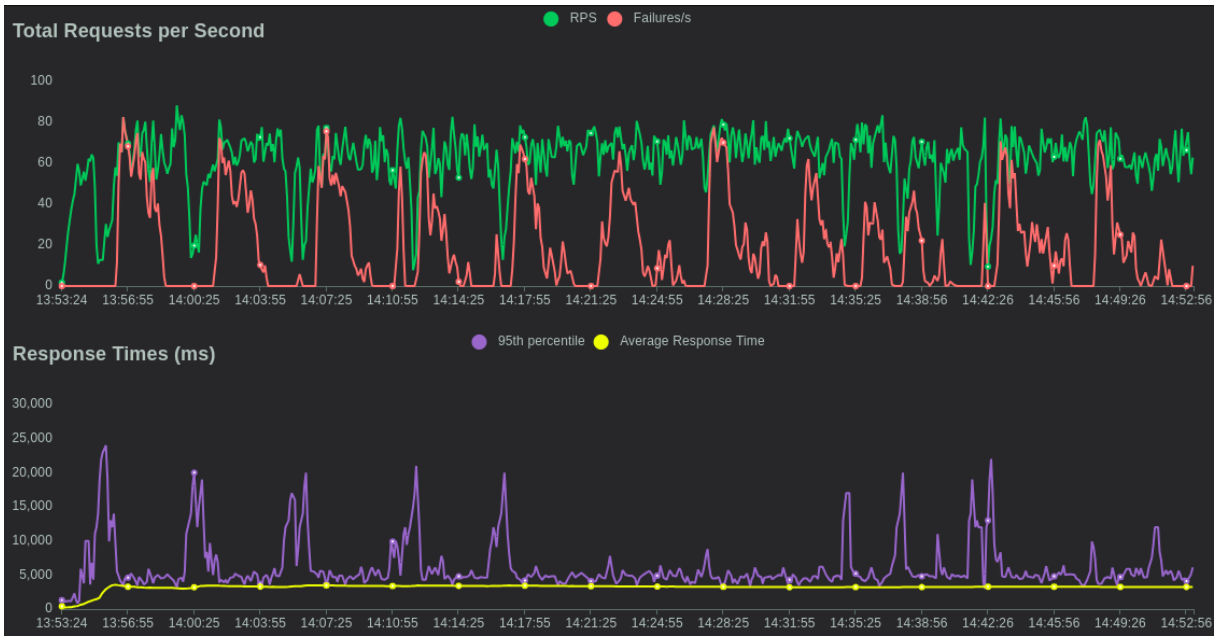
Fonte: Autor

Figura 26 – Avaliação A08: W2 300 Usuários - C2 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

Figura 27 – Avaliação A09: W2 300 Usuários - C3 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

médio de resposta foi de 3,38 segundos. O sistema também registrou um alto número de falhas 62.743, o que resultou em uma taxa significativa de 17,43 falhas por segundo, o que indica uma alta taxa de falhas, comprometendo a confiabilidade do sistema sob esta carga de trabalho (*workload*).

Ao usar a *API Gateway* desenvolvida (C2), houve uma redução no número total de requisições processadas para 208.769, com uma taxa de requisições por segundo de 57,99. O tempo médio de resposta com a *API Gateway* aumentou para 3,58 segundos, introduzindo um leve *overhead*. A taxa de falhas foi de 17,10 por segundo, levemente melhor do que no cenário sem *API Gateway* (C1), mas o serviço ainda apresenta um número considerável de falhas 61.556.

A Kong *API Gateway* (C3) demonstrou um desempenho semelhante ao do cenário sem *API Gateway* (C1) com uma taxa de 59,73 requisições por segundo, processando 215.038 requisições no total. O tempo médio de resposta foi de 3,25 segundos, o mais rápido entre os três cenários. No entanto, este cenário registrou o maior número de falhas 64.346, resultando na maior taxa de falhas por segundo 17,87. Embora a *Kong* (C3) seja eficiente em termos de velocidade de processamento, ele é menos confiável em comparação com os outros cenários.

A análise dos gráficos dos três cenários (Figuras 25, 26 e 27) mostra que o cenário com a *Kong* (C3) combina alta capacidade de processamento com tempos de resposta rápidos, demonstrando ser mais eficaz para o *endpoint mobile* sob carga de 300 usuários simultâneos, superando o cenário sem *API Gateway* (C1) e com a *API Gateway* desenvolvida (C2).

6.2.4 Avaliações com 300 Usuários Simultâneos - *Endpoint Webservice*

Examinamos o desempenho do *endpoint* `servidor` do portal de *Webservice* sob uma carga de 300 usuários simultâneos para os três cenários. As métrica são apresentadas na Tabela 12 e os gráficos ilustrando essas métricas são apresentados nas Figuras 28, 29 e 30.

Tabela 12 – Avaliações W2 300 Usuários Simultâneos - *Endpoint Webservice*

Avaliação	Cenário	Requisições	Falhas	Média(s)	Requisições(s)	Falhas(s)
A10	**C1**	609.633	0	0,24	169,34	0
A11	**C2**	570.710	53	0,36	158,53	0,01
A12	**C3**	600.449	0	0,27	166,79	0

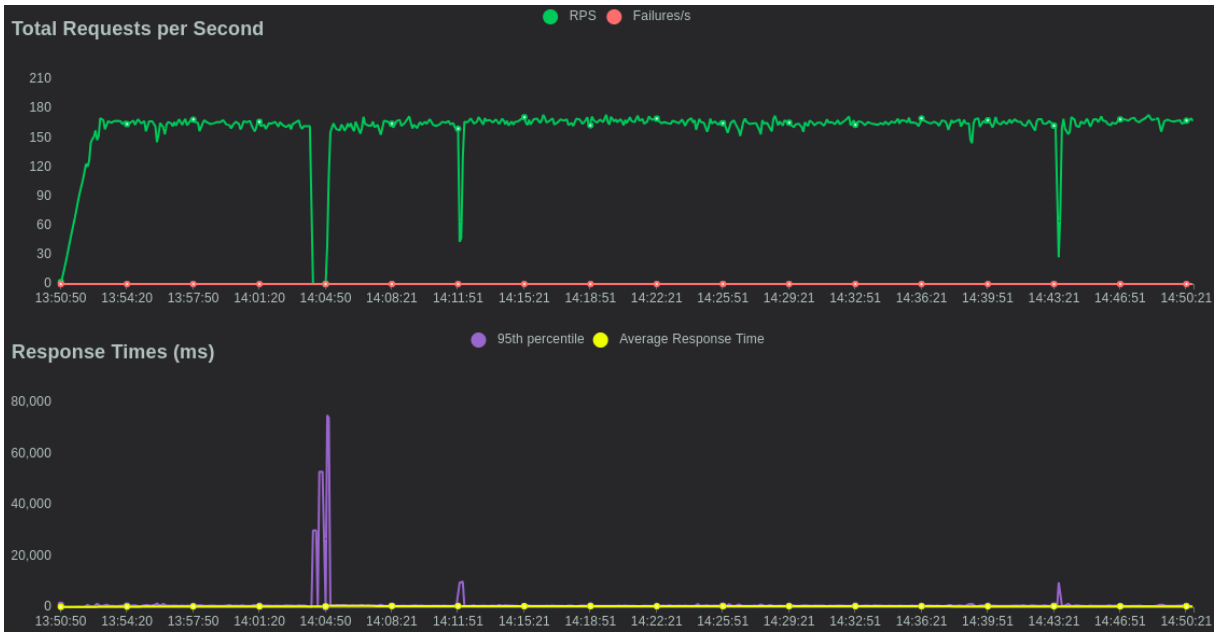
Figura 28 – Avaliação A10: W2 300 Usuários - C1 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo

Fonte: Autor

As requisições enviadas diretamente ao serviço do *Webservice* (C1), o sistema processou 609.633 requisições com uma taxa de 169.34 requisições por segundo. A latência média foi de 0,24 segundos, evidenciando uma resposta ágil. Não foram registradas falhas, o que demonstra alta confiabilidade do sistema quando não há intermediários.

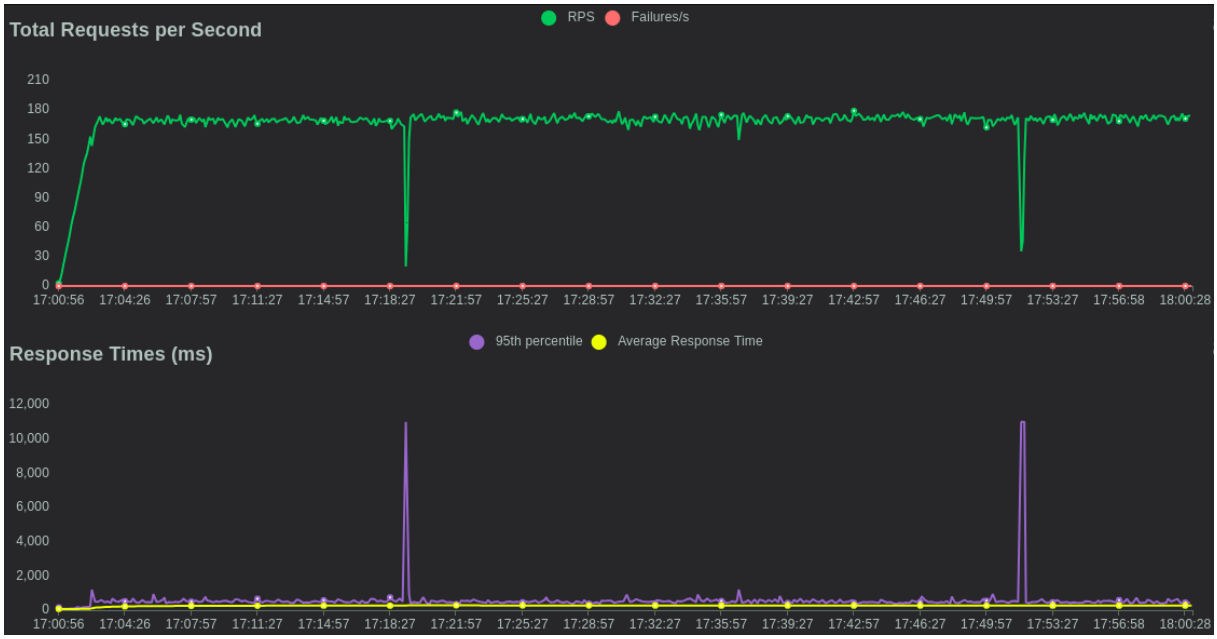
Com a utilização da *API Gateway* desenvolvida (C2), houve uma redução no número total de requisições processadas 570.710, resultando em uma taxa de 158.53 requisições por segundo. O tempo médio de resposta aumentou para 0,36 segundos, o que pode ser atribuído à sobrecarga introduzida pela *API Gateway*. Foram registradas 53 falhas, resultando em uma taxa de falhas por segundo de 0,01, indicando uma leve redução na confiabilidade.

Figura 29 – Avaliação A11: W2 300 Usuários - C2 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

Figura 30 – Avaliação A12: W2 300 Usuários - C3 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

A *Kong* (C3) mostrou-se eficiente, processando 600.449 requisições com uma taxa de 166.79 requisições por segundo. A latência média foi de 0,27 segundos. Sem falhas registradas, este cenário demonstrou alta confiabilidade, semelhante ao cenário sem *API Gateway* (C1).

A análise dos gráficos dos três cenários (Figuras 28, 29 e 30) para o *endpoint*

Webservice mostra que todos os cenários apresentam um desempenho muito semelhante, com uma pequena vantagem para o cenário sem *API Gateway* (C1) em termos de RPS e tempos de resposta. A introdução da *API Gateway* (C2 e C3) acrescenta um leve *overhead*, resultando em um pequeno aumento nos tempos de resposta.

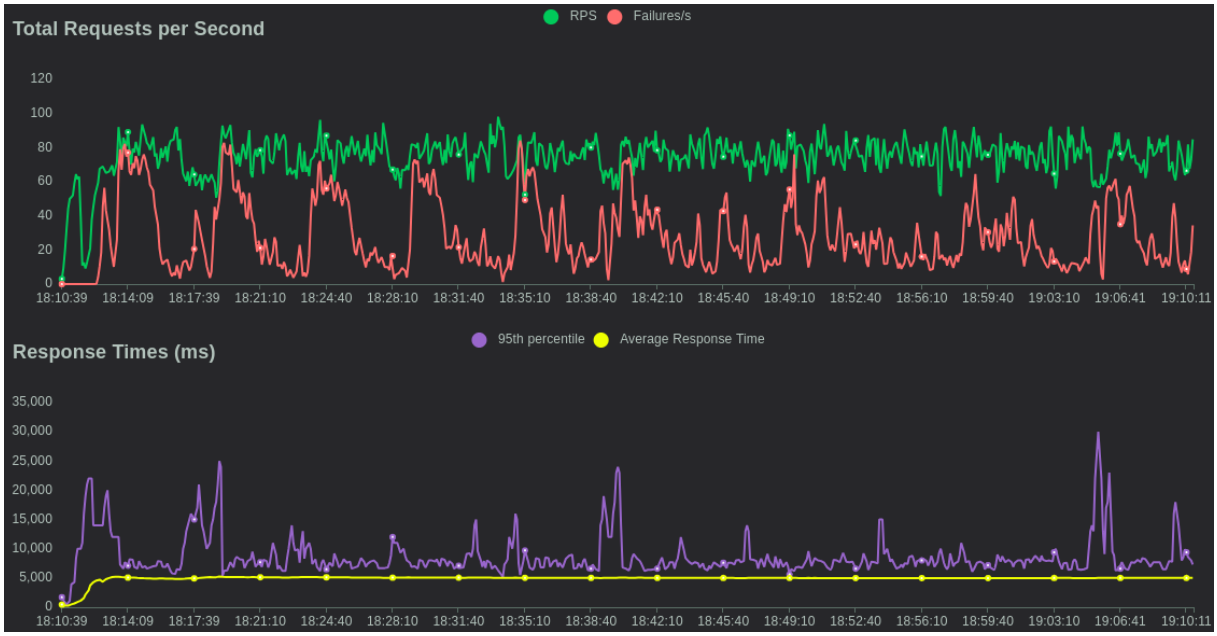
6.2.5 Avaliações com 500 Usuários Simultâneos - *Endpoint Mobile*

Realizamos uma análise detalhada do desempenho do *endpoint Mobile* sob uma carga de 500 usuários simultâneos para os três cenários e o *endpoint* que busca os horários de ônibus para o aplicativo *mobile* são exibidas na Tabela 13 e os gráficos ilustrando essas métricas são apresentados nas Figuras 31, 32 e 33.

Tabela 13 – Avaliações W3 500 Usuários Simultâneos - *Endpoint Mobile*

Avaliação	Cenário	Requisições	Falhas	Média(s)	Requisições(s)	Falhas(s)
A13	**C1**	271.439	102.459	5,01	75,40	28,46
A14	**C2**	261.794	119.097	5,25	72,72	33,08
A15	**C3**	266.643	112.615	5,12	74,07	31,28

Figura 31 – Avaliação A13: W3 500 Usuários - C1 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

As requisições feitas diretamente ao serviço sem a intermediação de uma *API Gateway* (C1), o sistema processou 271.439 requisições com uma taxa de 75,40 requisições por segundo. No entanto, o sistema registrou 102.459 falhas, resultando em uma taxa de falhas de 28,46 por segundo. O tempo médio de resposta foi de 5,01 segundos, indicando que, sob uma carga mais pesada, esse serviço apresenta problemas significativos de desempenho e confiabilidade.

Figura 32 – Avaliação A14: W3 500 Usuários - C2 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo

Fonte: Autor

Figura 33 – Avaliação A15: W3 500 Usuários - C3 - *Mobile* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo

Fonte: Autor

Utilizando a *API Gateway* desenvolvida (C2), houve uma pequena redução no número total de requisições processadas 261.794 e na taxa de requisições por segundo 72,72. O tempo médio de resposta aumentou para 5,25 segundos, indicando que a *API Gateway* introduz uma sobrecarga que impacta no desempenho. Além disso, foram registradas 119.097 falhas, resultando em uma taxa de falhas de 33,08 por segundo, o que indica uma

redução na confiabilidade do serviço ao utilizar esse *API Gateway* sob carga pesada.

A *Kong* (C3) processou 266.643 requisições, com uma taxa de 74,07 requisições por segundo. O tempo médio de resposta foi de 5,12 segundos, o que é ligeiramente melhor que a *API Gateway* desenvolvida (C2), mas ainda pior que o cenário sem *API Gateway* (C1). O número de falhas registradas foi de 112.615, resultando em uma taxa de falhas de 31,28 por segundo.

A análise dos gráficos dos três cenários (Figuras 31, 32 e 33) mostra que o cenário sem *API Gateway* (C1) apresenta o maior RPS e o menor tempo de resposta. O cenário com a Kong *API Gateway* (C3) é uma alternativa viável, com tempos de resposta rápidos e um RPS elevado. Já o cenário com a *API Gateway* desenvolvida (C2) não apresenta boas métricas de desempenho.

6.2.6 Avaliações com 500 Usuários Simultâneos - *Endpoint Webservice*

Vamos analisar as métricas da avaliação sob uma carga de trabalho de 500 usuários simultâneos para os três cenários e o *endpoint* que busca as informações de um determinado servidor são exibidas na Tabela 14 e os gráficos ilustrando essas métricas são apresentados nas Figuras 34, 35 e 36.

Tabela 14 – Avaliações W3 500 Usuários Simultâneos - *Endpoint Webservice*

Avaliação	Cenário	Requisições	Falhas	Média(s)	Requisições(s)	Falhas(s)
A16	**C1**	576.831	63	1,57	160,23	0,02
A17	**C2**	532.798	411	1,82	148,00	0,11
A18	**C3**	557.121	11	1,67	154,76	0

As requisições enviadas diretamente ao *Webservice* (C1), o sistema processou um total de 576.831 requisições, com uma taxa de 160,23 requisições por segundo. O tempo médio de resposta foi de 1,57 segundos. Apesar do número significativo de requisições, o sistema registrou 63 falhas, resultando em uma taxa de falhas de 0,02 por segundo, o que demonstra uma alta eficiência, mas com algumas instabilidades sob carga pesada.

O uso da *API Gateway* desenvolvida (C2) resultou em uma redução no número total de requisições processadas, com 532.798 requisições e uma taxa de 148,00 requisições por segundo. O tempo médio de resposta foi de 1,82 segundos, indicando uma sobrecarga adicional devido a *API Gateway*. Houve um aumento significativo no número de falhas, totalizando 411, o que levou a uma taxa de 0,11 falhas por segundo, indicando que, sob alta carga, a *API Gateway* pode impactar na confiabilidade do sistema.

A *Kong API Gateway* (C3) processou 557.121 requisições, com uma taxa de 154,76 requisições por segundo, demonstrando um bom equilíbrio entre eficiência e velocidade. O tempo médio de resposta foi de 1,67 segundos, o que é intermediário em comparação aos outros cenários. O número de falhas foi muito baixo, com apenas 11 registradas,

Figura 34 – Avaliação A16: W3 500 Usuários - C1 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo

Fonte: Autor

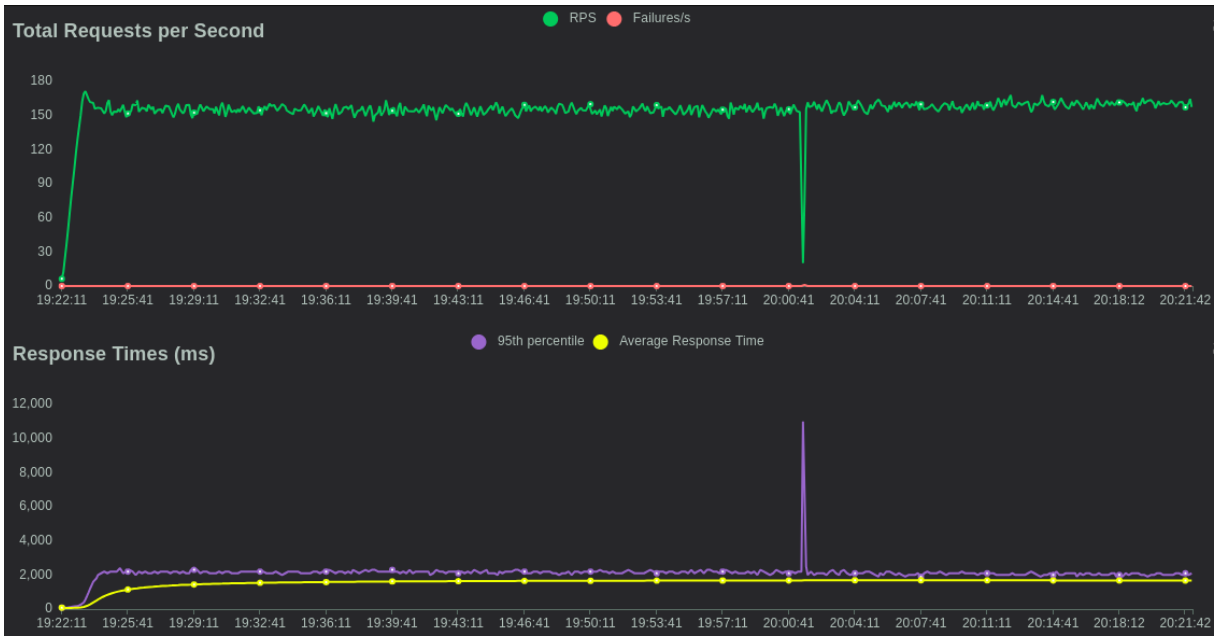
Figura 35 – Avaliação A17: W3 500 Usuários - C2 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo

Fonte: Autor

resultando em uma taxa de falhas por segundo próxima de zero, indicando que a Kong *API Gateway* oferece um bom desempenho.

A análise dos gráficos dos três cenários (Figuras 34, 35 e 36) mostra que o cenário sem *API Gateway* (C1) oferece o melhor desempenho em termos de RPS e o menor tempo

Figura 36 – Avaliação A18: W3 500 Usuários - C3 - *Webservice* (a) Requisições Totais por Segundo (b) Tempos Reposta por Milissegundo



Fonte: Autor

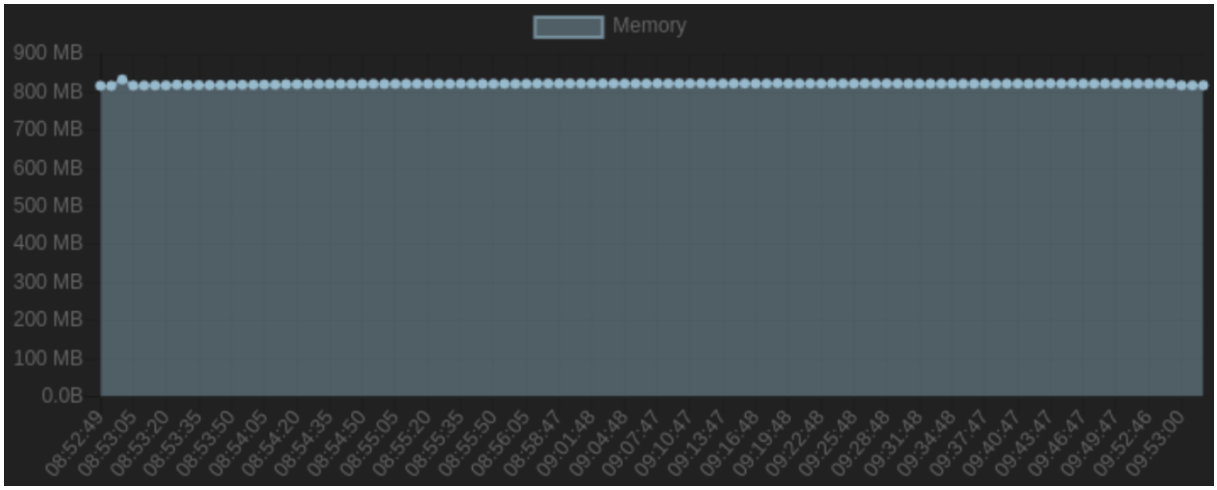
de resposta. O cenário com a Kong *API Gateway* (C3) apresenta um desempenho quase equivalente ao C1, com um pequeno aumento no tempo de resposta. Já o cenário com a *API Gateway* desenvolvida (C2) compromete tanto o desempenho em RPS quanto o tempo de resposta.

6.3 Consumo de Recursos Computacionais

Além das métricas de desempenho relacionadas aos acessos aos *endpoints*, como `Número Total de Requisições`, `Número Total de Falhas`, `Tempo Médio de Resposta`, `Taxa de Requisições por Segundo` e a `Taxa de Falhas por Segundo`, também foram coletadas métricas de consumo computacional para avaliar o uso de recursos, tais como: `Memória`, `Processamento` e `Tráfego de Rede`. Nesta seção, realizamos uma análise comparativa com base nos dados coletados com o objetivo de responder à **QP4.** (Há diferenças significativas no consumo de recursos computacionais entre as soluções testadas?), a fim de avaliar o custo de processamento computacional entre o cenário com a *API Gateway* desenvolvida (C2) e o cenário com a Kong API Gateway (C3).

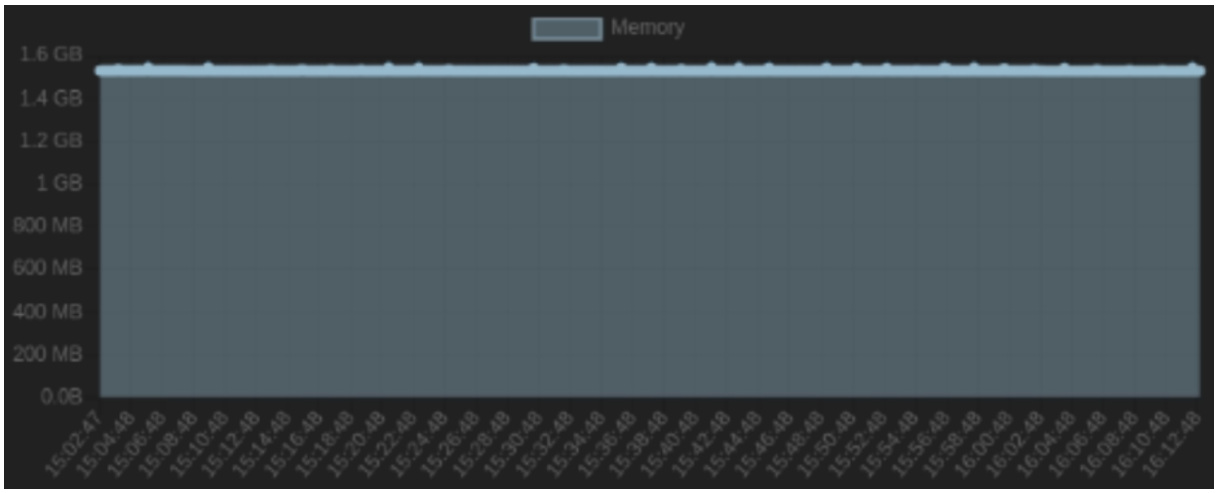
Em termos de consumo de memória, a *API Gateway* desenvolvida (C2) demonstrou ser mais eficiente, utilizando cerca de 50% menos memória que a Kong (C3), como mostrado nas Figuras 37 e 38. Esse comportamento se manteve em todas as cargas de trabalho testadas (W1, W2 e W3). Essa diferença representa uma vantagem significativa na utilização de recursos de hardware, permitindo um uso mais eficiente e econômico da memória disponível.

Figura 37 – W3 500 Usuários - C2 - *Mobile* - Utilização de Memória



Fonte: Autor

Figura 38 – W3 500 Usuários - C3 - *Mobile* - Utilização de Memória



Fonte: Autor

Quanto ao consumo de *Central Processing Unit* (CPU) nos testes realizados com a carga de trabalho de 500 usuários (W3) para o *endpoint* `Mobile`, observamos comportamentos distintos entre a *API Gateway* desenvolvida (C2) e a *Kong* (C3). A *API Gateway* desenvolvida (C2) apresentou picos de CPU significativos na fase inicial de execução do teste, chegando na casa dos 120% por breves períodos. Após esse período inicial, o consumo de CPU estabilizou em níveis mais baixos, torno de 15%, com oscilações esporádicas de 40% a 60%, indicando uma utilização eficiente dos recursos. Por outro lado, a *Kong* (C3) mostrou um padrão diferente. Em vez de picos concentrados no início, o consumo de CPU se manteve estável em cerca de 10%, com flutuações regulares e picos constantes chegando a 55%, distribuídos ao longo de toda a execução do teste.

Embora a *API Gateway* desenvolvida consuma mais CPU no início da execução, a *Kong* apresenta um padrão de consumo distribuído e constante ao processar as requisições

de 500 usuários simultâneos para o *endpoint mobile*. Ambos os cenários demonstram boa eficiência, porém com abordagens distintas para a gestão da carga e alocação de recursos.

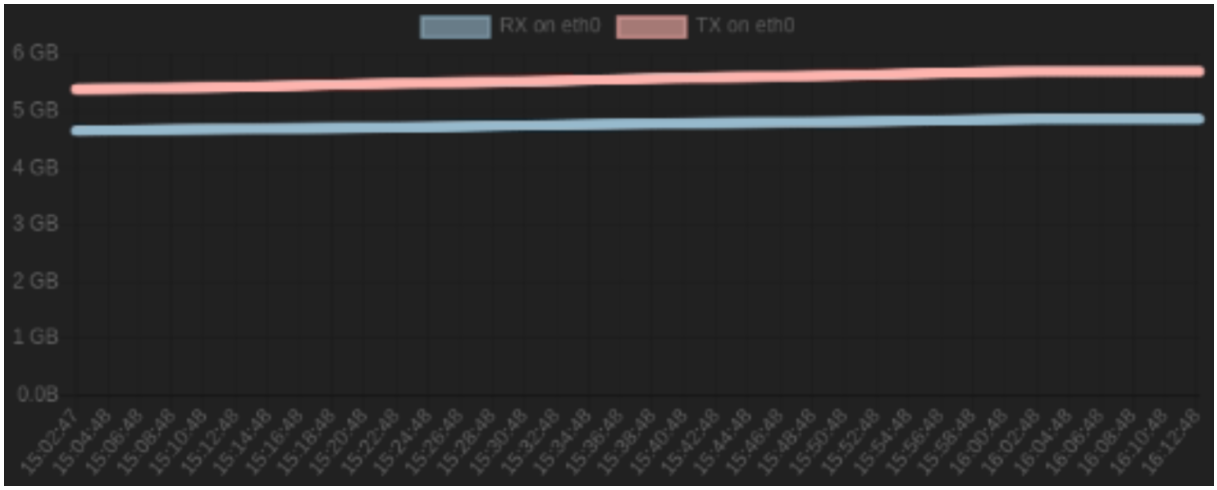
No cenário (C2), para o *endpoint* `Webservice` com o *workload* (W3), o consumo de CPU apresentou picos de até 200%, concentrados na fase inicial do processamento. Após os picos iniciais, o consumo de CPU se estabilizou em um nível mais baixo, em torno de 30%, com pequenas variações, sem voltar a atingir os picos anteriores de 200%. O comportamento no cenário (C3) é diferente em relação ao (C2). O uso de CPU varia constantemente entre 15% e 65%, com picos regulares, distribuição equilibrada e contínua do processamento ao longo do tempo.

A *API Gateway* desenvolvida (C2) estabiliza o consumo de CPU rapidamente após os picos iniciais, enquanto a *Kong* (C3) apresenta um padrão de consumo mais oscilante e constante.

No quesito tráfego de rede, a *Kong* (C3) apresentou uma vantagem clara, consumindo, significativamente menos tráfego de rede, tanto no recebimento quanto no envio de dados, em comparação com a *API Gateway* desenvolvida (C2).

Para o *endpoint Mobile* sob (W3), a *Kong* (C3) teve um tráfego médio aproximado de rede de 4,9 GB de dados recebido (RX) e 5,5 GB enviado (TX), conforme Figura 39, comparado aos 13,15 GB recebido (RX) e 10 GB enviado (TX) pela *API Gateway* desenvolvida (C2), conforme Figura 40.

Figura 39 – W3 500 Usuários - C3 - *Mobile* - Utilização de *Network*

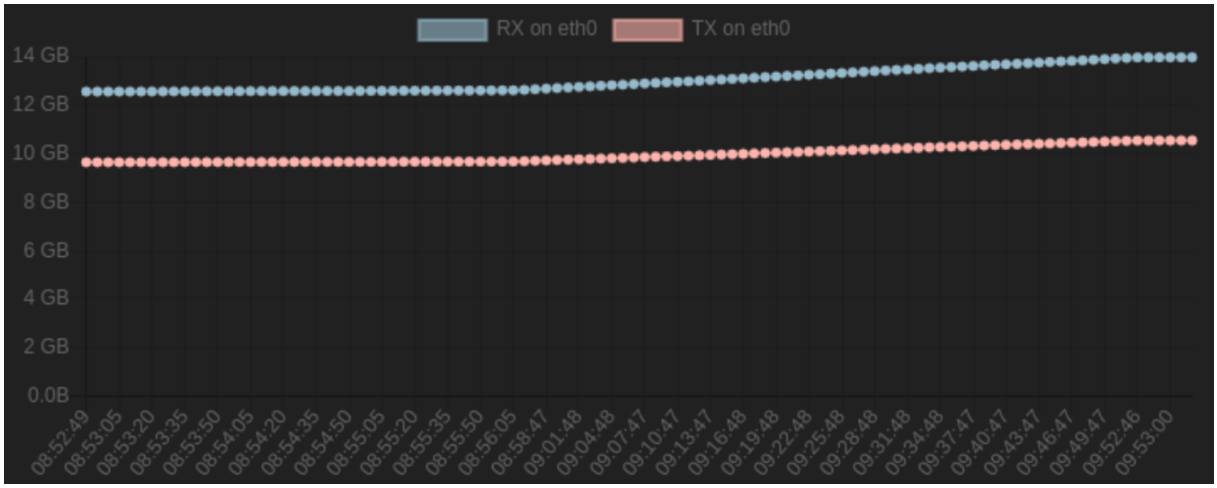


Fonte: Autor

No *endpoint Webservice* sob (W3), com a *Kong* consumindo um tráfego médio aproximado de rede de 5,9 GB de dados recebidos (RX) e 7 GB enviados (TX), conforme Figura 41, comparado aos 16,1 GB recebidos (RX) e 12,1 GB enviados (TX) pela *API Gateway* desenvolvida (C2), conforme Figura 42.

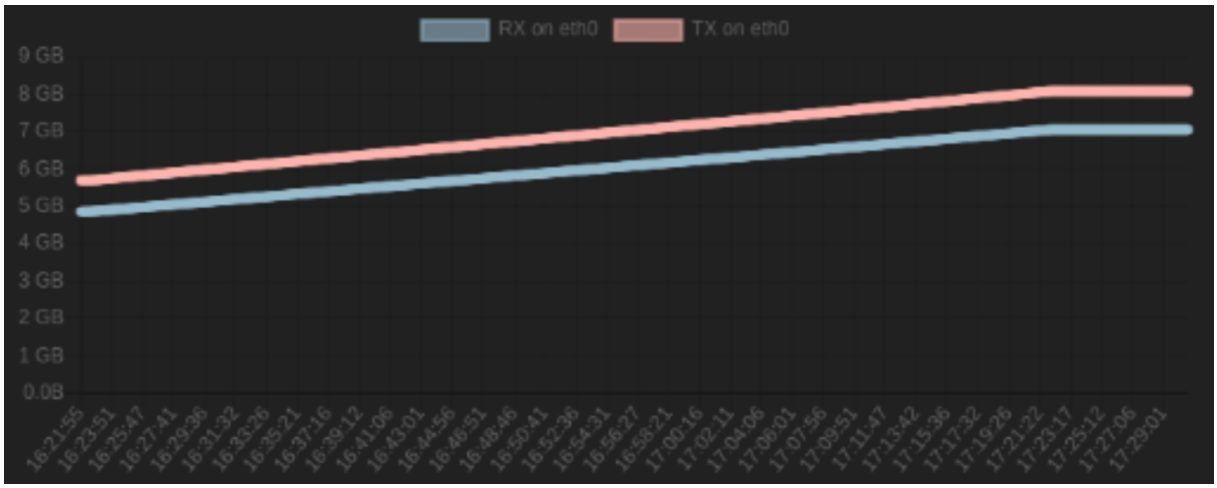
Em resumo, a análise comparativa entre os cenários C2 (*API Gateway* Desenvolvida) e C3 (*Kong*) revela que a solução desenvolvida se destaca em termos de consumo de memória, enquanto a *Kong* apresenta melhor desempenho no uso de rede, especialmente

Figura 40 – W3 500 Usuários - C2 - *Mobile* - Utilização de *Network*



Fonte: Autor

Figura 41 – W3 500 Usuários - C3 - *Webservice* - Utilização de *Network*

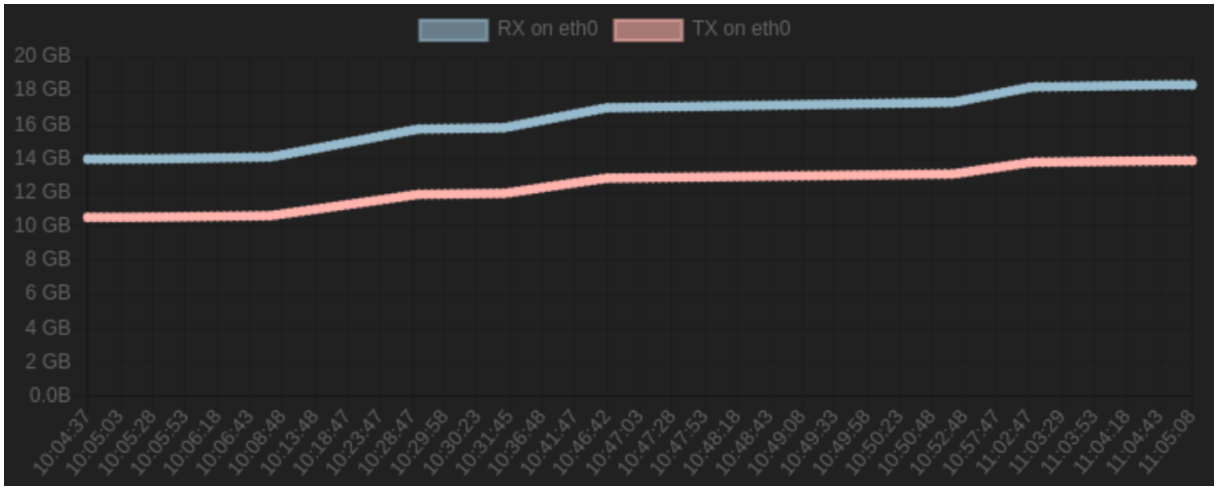


Fonte: Autor

sob cargas de trabalho mais pesadas, supõe-se que seja devido à compressão de dados, a qual reduz o tráfego de dados. O consumo de CPU se manteve equilibrado entre as duas soluções, sugerindo que ambas as *API Gateways* são eficientes no processamento das requisições, com a *Kong* tendo uma leve vantagem em cenários de alta demanda. Essas observações fornecem *insights* valiosos para a escolha da *API Gateway* a ser utilizado, dependendo das prioridades de recursos computacionais e da infraestrutura disponível.

As métricas de consumo de recursos computacionais foram coletadas diretamente do contêiner de cada *API Gateway* (C2 e C3) utilizando o *Portainer Community Edition*, versão 2.20.3. O *Portainer* é uma ferramenta de código aberto que oferece uma maneira simples e segura de gerenciar contêineres *Docker*. Para cada contêiner, o *Portainer* fornece a funcionalidade "`Stats`", que mostra as estatísticas de utilização de "`Memória`", "`Processamento`", "`Network`" e de "`I/O`" em tempo real sobre o contêiner.

Figura 42 – W3 500 Usuários - C2 - *Webservice* - Utilização de *Network*



Fonte: Autor

6.4 Discussão dos Resultados

6.4.1 *Endpoint Mobile*

O cenário sem *API Gateway* (C1) geralmente processa o maior número de requisições por segundo, mas isso vem acompanhado de uma alta taxa de falhas, especialmente quando a carga aumenta (W2 e W3). Embora normalmente apresente o menor tempo de resposta, a confiabilidade é comprometida pelo elevado número de falhas.

Nos cenários com *API Gateway* (C2 e C3), observa-se uma leve redução no RPS devido ao *overhead* introduzido, mas os problemas de falhas persistem. A *Kong API Gateway* (C3) oferece um RPS um pouco maior que a *API Gateway* desenvolvida (C2), mas também sofre com um número elevado de falhas. Já o cenário com a *API Gateway* desenvolvida (C2) introduz uma sobrecarga que aumenta o tempo de resposta, mas oferece um equilíbrio um pouco melhor entre desempenho e confiabilidade, embora as falhas ainda sejam significativas. O cenário com a *Kong* (C3) apresenta um tempo de resposta próximo ao de C1, mas com uma taxa de falhas similar ou até superior, especialmente sob cargas mais pesadas.

O *endpoint mobile* enfrenta dificuldades significativas de confiabilidade em todos os cenários e *workloads*, independentemente do uso de *API Gateway* (C2 e C3). As altas taxas de falhas registradas sugerem que o problema está diretamente relacionado ao *endpoint* do próprio serviço, e não aos cenários (C2 e C3) configurados com uma *API Gateway*. Se o serviço não consegue lidar eficientemente com um volume elevado de requisições, ele pode começar a falhar sob carga intensa. Uma revisão da capacidade do serviço e uma análise aprofundada das operações internas no serviço do *endpoint mobile* são recomendadas para mitigar essas falhas e melhorar a confiabilidade geral.

6.4.2 *Endpoint Webservice*

O cenário sem *API Gateway* (C1) apresenta o maior RPS sem registrar falhas, demonstrando alta eficiência e oferecendo o menor tempo de resposta, sem comprometer a confiabilidade. Os cenários com *API Gateway* (C2 e C3) apresentam uma pequena redução no RPS e um pequeno aumento no tempo de resposta, provavelmente devido ao *overhead*, mas mantêm a confiabilidade, com poucas ou nenhuma falha registrada. O tempo de resposta permanece dentro de um intervalo aceitável.

O uso de uma *API Gateway* (C2 e C3) é justificado devido a necessidade de introduzir um controle adicional ou funcionalidades específicas na arquitetura de software proposta. Ambos os cenários de *API Gateway* (C2 e C3) mantêm um bom desempenho, confiabilidade e mostram-se adequados para ambientes de produção.

As próximas discussões dizem respeito ao contexto geral de todos os *workloads* e cenários. A quantidade de requisições em W2 aumentou significativamente em comparação a W1, mas em W3 houve um decréscimo em relação a W2 para o *endpoint Webservice*, já para o *endpoint mobile* o aumento na proporcionalidade não se manteve comparando o aumento de W1 para W2. A redução na quantidade de requisições observada em W3 pode indicar uma saturação do sistema, uma vez que as requisições são direcionadas constantemente para o mesmo *endpoint*.

6.4.3 Respostas às Questões de Pesquisa

Sintetizamos as respostas às questões de pesquisa (QPs) descritas no início do relato experimental e apresentadas nas Figuras 43, 44, 45 e 46.

De forma geral, a *Kong API Gateway* (C3) apresentou tempos médios de resposta menores em comparação com a *API Gateway* desenvolvida (C2). Contudo, para 500 usuários simultâneos, ambas as ferramentas registraram aumento no tempo de resposta, com a *Kong* mantendo uma ligeira vantagem. Já o acesso direto ao serviço (C1) apresentou os menores tempos de resposta, mas enfrentou problemas significativos de confiabilidade em todos os *workloads* no *endpoint Mobile*.

Figura 43 – Resposta à QP1.

O acesso direto ao serviço (C1) alcançou o maior número de requisições por segundo (RPS) em todas as cargas de trabalho. A *Kong* (C3) apresentou desempenho próximo ao C1, superando a *API Gateway* desenvolvida (C2). Esse padrão foi consistente em todas as análises, independentemente da carga de trabalho ou do *endpoints*.

Figura 44 – Resposta à QP2.

Para o *endpoint Webservice*, a *Kong* (C3) e a *API Gateway* desenvolvida (C2) demonstraram uma baixa taxa de falhas, com valores quase inexistentes, indicando alta confiabilidade. No entanto, para o *endpoint Mobile*, todas as soluções (C1, C2 e C3) apresentaram taxas de falhas significativas, que aumentaram proporcionalmente à carga de trabalho, sugerindo limitações inerentes ao serviço desse *endpoint*.

Figura 45 – Resposta à QP3.

Sim, houve diferenças relevantes:

Consumo de Memória: A *API Gateway* desenvolvida (C2) foi mais eficiente, consumindo aproximadamente 50% menos memória que a *Kong* (C3);

Uso de CPU: A *API Gateway* desenvolvida (C2) apresentou picos iniciais mais elevados de CPU, mas estabilizou em níveis mais baixos durante a execução. Já a *Kong* manteve um padrão de uso mais constante, porém com um consumo médio maior.

Tráfego de Rede: A Kong (C3) mostrou-se mais eficiente, consumindo menos largura de banda devido a técnicas de compressão de dados.

Figura 46 – Resposta à QP4.

6.5 Ameaças à Validade da Avaliação

Apresentamos as principais ameaças que poderiam comprometer a validade da avaliação experimental (WOHLIN et al., 2012):

Validade de Construção: Refere-se à relação entre o que foi medido no experimento e o que se pretende avaliar. Erros podem surgir devido à escolha inadequada de métricas ou interpretações ambíguas. Por exemplo, a definição incorreta de métricas, como tempo de resposta, RPS e falhas, pode não capturar completamente os objetivos do estudo. Para mitigar essa ameaça, garantimos que as métricas selecionadas estejam alinhadas com os objetivos da pesquisa e sejam amplamente reconhecidas pela comunidade científica. Possíveis imprecisões nos instrumentos de coleta de dados, como *logs* do servidor ou ferramentas de monitoramento, também representam uma ameaça. Para mitigá-las, utilizamos ferramentas validadas e confiáveis, além de calibrar os instrumentos antes do experimento. Adicionalmente, o design experimental foi submetido a uma revisão por pares, visando identificar ambiguidades e aumentar a confiabilidade dos resultados;

Validade Interna: Refere-se à possibilidade de fatores externos ou não controlados influenciarem os resultados do experimento. A concorrência por recursos entre processos executados no mesmo ambiente pode causar interferências. Para evitar essa concorrência, isolamos cada cenário de execução em contêineres *Docker* independentes, garantindo que todos os experimentos sejam realizados em

um ambiente controlado e isolado. Além disso, erros operacionais durante os testes, causados por fatores humanos, podem afetar a validade interna. Para mitigar essa ameaça, realizamos testes de fumaça (*Smoke Testing*) para verificar as funcionalidades básicas do ambiente a ser testado;

Validade Externa: Trata da generalização dos resultados obtidos no experimento para outros contextos ou cenários. Os resultados podem ser específicos para as ferramentas testadas ou o ambiente utilizado, limitando o escopo experimental. Além disso, cenários de carga de trabalho podem ser artificiais e não refletir condições reais de uso em produção. Testar apenas um conjunto limitado de *endpoints* pode não representar outros cenários, reduzindo a diversidade nos *endpoints* testados. Para mitigar essas ameaças, buscamos testar as ferramentas em diferentes configurações e com cargas de trabalho variadas, simulando a diversidade de cenários reais. Além disso, comparamos os resultados com ferramentas amplamente reconhecidas no campo para validar a generalização;

Validade de Conclusão: Refere-se à confiabilidade das conclusões tiradas a partir dos dados coletados. O número de execuções pode não ser suficiente para garantir resultados estatisticamente significativos. Para mitigar essa ameaça, asseguramos que os experimentos sejam repetidos várias vezes, a fim de coletar uma quantidade suficiente de dados. Além disso, a interpretação subjetiva dos resultados pode representar outra ameaça. Para minimizar essa influência, submetemos os resultados a uma revisão por pares, com o objetivo de identificar possíveis erros ou vieses na análise.

6.6 Resumo do Capítulo

Neste capítulo, exploramos diferentes aspectos da avaliação de desempenho dos sistemas testados, com foco nos testes de carga realizados em diferentes cenários e *endpoints*. Iniciamos apresentando o *Locust*, ferramenta utilizada para simular múltiplos usuários acessando os serviços simultaneamente. Através dela, pudemos coletar métricas de desempenho, como tempo de resposta, taxa de sucesso e falha, e o número de requisições por segundo. Em seguida, discutimos os dois serviços avaliados, o *endpoint* de informações do Servidor, amplamente utilizado por diversas aplicações institucionais, e o *endpoint* de Horários de Ônibus, voltado especialmente para dispositivos móveis. Também detalhamos os três cenários de teste utilizados. Cada cenário foi avaliado com diferentes cargas de trabalho para medir a eficácia da utilização de uma *API Gateway*.

Explicamos como os testes foram estruturados com diferentes cargas de trabalho, e como os resultados foram organizados para facilitar a análise dos principais indicadores de desempenho. Ao longo das avaliações, observamos que, com o aumento do número de usuários, as falhas se tornaram mais evidentes no *endpoint Mobile*, enquanto o *endpoint Webservice* apresentou maior estabilidade, principalmente nos cenários sem o uso de uma

API Gateway. Além disso, analisamos o consumo de recursos computacionais das *API Gateways*, como memória, CPU e tráfego de rede em cada cenário. A *API Gateway* desenvolvida mostrou-se mais eficiente em termos de consumo de memória, enquanto a *Kong* destacou-se pelo uso otimizado de rede. Ambas as *API Gateway* tiveram utilização de processamento de forma equilibrada.

Discutimos os principais resultados da análise, enfatizando o comportamento inconsistente do *endpoint Mobile* em todos os cenários. Também ressaltamos que, embora o uso de uma *API Gateway* introduza um *overhead*, ela pode trazer benefícios importantes, como maior controle de acesso e confiabilidade.

Por fim, sintetizamos as respostas às questões de pesquisa e destacamos as principais ameaças que poderiam comprometer a validade da avaliação.

7 CONSIDERAÇÕES FINAIS

No cenário em constante evolução do desenvolvimento de software, a arquitetura de microsserviços tornou-se a base para a construção de sistemas modernos, robustos e escaláveis. Uma *API Gateway*, como componente central, desempenha um papel fundamental na comunicação, aprimorando o controle de acesso de um ecossistema com base em microsserviços.

A *API Gateway* foi projetada para centralizar as requisições externas aos webservices do SIE, com o objetivo de proteger o acesso contra usuários não autorizados. Ela oferece uma camada adicional de controle de acesso, garantindo uma comunicação mais segura e eficiente entre os sistemas da UFSM.

A implementação da *API Gateway* demonstrou que é possível melhorar significativamente o controle de acesso aos *webservices*, oferecendo uma solução moderna e eficaz para ajudar a mitigar vulnerabilidades. A *API Gateway* visa garantir a validação de *tokens* de autenticação e integra funcionalidades de roteamento e coleta de *logs* para monitoramento.

As avaliações de desempenho realizadas por meio de diferentes cenários, *workloads* e *endpoints* permitiram uma análise comprativa entre a *API Gateway* desenvolvida e a ferramenta *Kong*. As avaliações de desempenho mostraram que o uso de uma *API Gateway* se justifica nessa arquitetura, pois não causou impacto significativo no tempo de resposta das requisições e traz vantagens como controle de acesso, roteamento e registro de *logs*. O tempo de resposta médio e a taxa de requisições por segundo mantiveram-se dentro do aceitável, mesmo sob cargas elevadas de usuários simultâneos. A análise comparativa entre a *API Gateway* desenvolvida e a *Kong API Gateway* mostrou que ambas são soluções adequadas, cada uma com suas vantagens e desvantagens em termos de desempenho e consumo de recursos.

Esse trabalho trouxe contribuições importantes para o controle de acesso dos *webservices* do SIE. A centralização do controle de acesso e a validação de *tokens* permitem um gerenciamento mais eficiente das requisições aos *webservices*. A funcionalidade de geração de *logs* possibilita uma visão do comportamento das requisições e um monitoramento eficaz, garantindo maior rastreabilidade e controle sobre o uso dos serviços. Além disso, o projeto contribui diretamente para a modernização da arquitetura de sistemas da UFSM.

A *API Gateway* desenvolvida demonstrou sua capacidade de atender aos requisitos de desempenho necessários para a arquitetura dos *webservices* do SIE. O desenvolvimento da *API Gateway* trouxe diversas oportunidades para trabalhos futuros. Como recomendações para a Divisão de Análise e Desenvolvimento de Sistemas do CPD, sugerimos a adoção e continuidade do desenvolvimento da *API Gateway*. A tecnologia utilizada na implementação da *API Gateway* é familiar da equipe de desenvolvimento do CPD, isso torna mais fácil o aprimoramento e a adição de novas funcionalidades e customizações.

Entre os possíveis aprimoramentos, destaca-se a implementação de políticas para *rate-limiting*, restrição de requisições por endereço IP, *cache* das requisições e a evolução na coleta de *logs*. Essas funcionalidade poderão melhorar o desempenho, aumentar a robustez e permitir uma gestão ainda mais eficiente.

Em termos de segurança, houve melhorias, pois agora há validação de *token* para acessar os *webservices*. Quanto ao *overhead*, ele não impactou no desempenho das aplicações. Este trabalho teve como foco a avaliação de desempenho, adiando a análise de segurança para estudos futuros. Portanto, propomos a realização de experimentos voltados à avaliação da segurança dos *webservices* do SIE, a fim de verificar a eficácia da *API Gateway* na proteção contra acessos não autorizados. Neste sentido, sugerimos ao Departamento de Desenvolvimento e ao Setor de Segurança do CPD que, à medida que o controle de acesso aos *webservices* do SIE for operacionalizado, seja feita uma nova solicitação ao CISC para refazer o teste de segurança *white-box* realizado anteriormente, a fim de verificar se as vulnerabilidades foram resolvidas após a introdução de uma *API Gateway* na arquitetura.

A descrição da arquitetura, ferramentas e tecnologias utilizadas, assim como o processo de desenvolvimento, está disponível publicamente nesta dissertação. No entanto, por se tratar de um caso real em uma instituição pública, o acesso ao código-fonte da *API Gateway* está restrito ao time de desenvolvimento do CPD.

O desenvolvimento e a avaliação da *API Gateway* trouxeram aprendizados valiosos. A experiência demonstrou a importância de uma arquitetura bem planejada e a escolha adequada de tecnologias, para garantir a eficiência do sistema.

A *API Gateway* desenvolvida para os *webservices* do SIE da UFSM cumpriu com os objetivos estabelecidos, oferecendo uma solução moderna, robusta e capaz de atender às demandas de controle de acesso e ajudar a mitigar os de riscos de acesso não autorizado. A comparação com a *Kong API Gateway* mostrou que ambas as soluções são viáveis, cada uma com suas vantagens específicas. No entanto, a solução desenvolvida se destaca por proporcionar maior personalização e controle. Assim, além de atender às necessidades dos sistemas da Instituição, este trabalho serve como uma referência para outras instituições que buscam modernizar seus sistemas e aprimorar a gestão de acesso em arquiteturas de microsserviços, alinhando-se às exigências de conformidade e proteção de dados.

Esta dissertação oferece contribuições importantes ao propor uma solução de *API Gateway* que atende aos desafios atuais de controle de acesso na arquitetura de microsserviços, com potencial para evoluir e atender às futuras necessidades da UFSM.

Além disso, ressalta-se que durante o período de estudos submetemos alguns artigos a diversos eventos, alguns associados diretamente ao tema do projeto de pesquisa, outros correlacionados à outros projetos de pesquisa conduzidos pelo professor orientador.

A seguir listamos algumas publicações:

Artigos Submetidos e Rejeitados

1. *A Comprehensive Analysis of Authentication Solutions for Microservices: A Systematic Review* - International Conference on Utility and Cloud Computing (UCC'23) - Rejeitado;
2. *Exploring Authentication Solutions in Microservices: A Comprehensive Review - Technical Track on Software Architecture: Theory, Technology, and Applications* (SA-TTA'23) - Rejeitado;

Artigo Aceito para Publicação

1. Avaliação de *API Gateways* para o Gerenciamento de Acessos à Microsserviços - VIII Escola Regional de Engenharia de Software (ERES'24) - Aceito.

Artigos Aceitos e Publicados

1. CARGNELUTTI, Rodrigo; BERNARDINO, Maicon; GARCIA, Renato; SILVA, Williamson. Um Estudo Exploratório sobre o uso do ChatGPT na Melhoria e Revisão da Escrita de Artigos Científicos. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (SBIE'23), 34., 2023, Passo Fundo/RS, 2023 p. 1271–1281;
2. BERNARDINO, Maicon; CARGNELUTTI, Rodrigo; GARCIA, Renato; SILVA, Williamson. *The Use of ChatGPT in Improving and Reviewing Scientific Paper Writing: An Exploratory Study.* in IEEE Revista Iberoamericana de Tecnologias del Aprendizaje (IEEE-RITA), vol. 19, pp. 120-128, 2024.
3. CARGNELUTTI, Rodrigo; BASSO, Fábio Paulo; BERNARDINO, Maicon. Uma Análise Abrangente de Soluções de Autenticação para Microsserviços: Uma Revisão Sistemática. In: VII Escola Regional de Engenharia de Software (ERES'23), 7., 2023, Maringá/PR. pp. 228–237.

Prêmio

1. O Comitê Organizador da VII ESCOLA REGIONAL DE ENGENHARIA DE SOFTWARE em conjunto com a Coordenação do Fórum de Pós-Graduação concede o **PRÊMIO de MELHOR ARTIGO DO FÓRUM DE PÓS-GRADUAÇÃO** ao trabalho intitulado Uma Análise Abrangente de Soluções de Autenticação para Microsserviços: Uma Revisão Sistemática de autoria de Rodrigo Cargnelutti (Unipampa), Maicon Bernardino (Unipampa) e Fábio Basso (Unipampa). Maringá-PR, dezembro de 2023.

Publicações Futuras

1. Workshop de Tecnologia da Informação e Comunicação das Instituições Federais de Ensino Superior do Brasil (WTICIFES'25).
2. Technical Track on Software Architecture: Theory, Technology, and Applications (SA-TTA'25).

ANEXO A – AUTORIZAÇÃO CPD/UFSM

Ministério da Educação
Universidade Federal de Santa Maria
Centro de Processamento de Dados

Santa Maria, 25 de junho de 2024.

AUTORIZAÇÃO

O Centro de Processamento de Dados, como órgão central de TI da UFSM, apoia a realização de trabalhos de pesquisa envolvendo aplicação de casos na Instituição. Dessa forma, apoia e autoriza o trabalho realizado pelo Técnico de Tecnologia da Informação Rodrigo Cargnelutti, lotado no CPD sob matrícula SIAPE 1215456, intitulado *Modernização da Arquitetura de Software e Segurança para os Sistemas da UFSM: Implementação de um API Gateway*.

gov.br
Documento assinado digitalmente
GUSTAVO ZANINI KANTORSKI
Data: 25/06/2024 17:14:15-0300
Verifique em https://validar.iti.gov.br

Analista de TI Gustavo Zanini Kantorski

Diretor do CPD/UFSM

REFERÊNCIAS

AGILAR, E. de V. **Uma Abordagem Orientada a Serviços para a Modernização de Sistemas Legados**. 2016. Disponível em: <http://icts.unb.br/jspui/handle/10482/22250>. Citado na página 15.

ALENCAR, J. et al. Comparação de desempenho entre soluções de interoperabilidade. In: **Anais do X Workshop de Computação Aplicada em Governo Eletrônico**. Porto Alegre, RS, Brasil: SBC, 2022. p. 25–36. ISSN 2763-8723. Disponível em: <https://sol.sbc.org.br/index.php/wcge/article/view/20708>. Citado 3 vezes nas páginas 25, 27 e 37.

ALMEIDA, M. G. de; CANEDO, E. D. Authentication and Authorization in Microservices Architecture: A Systematic Literature Review. **Applied Sciences**, v. 12, n. 6, 2022. ISSN 2076–3417. Citado 4 vezes nas páginas 23, 29, 38 e 43.

ALSHUQAYRAN, N.; ALI, N.; EVANS, R. A systematic mapping study in microservice architecture. In: **9th International Conference on Service-Oriented Computing and Applications (SOCA)**. Macau, China: IEEE, 2016. p. 44–51. Citado 2 vezes nas páginas 29 e 43.

ARAUJO, L.; MARINHO, E. Desafios da autenticação e autorização na comunicação entre serviços em arquiteturas de microsserviços. 09 2023. Citado na página 39.

AWATI, R.; WIGMORE, I. **Monolithic Architecture**. 2023. Disponível em: <https://www.techtarget.com/whatis/definition/monolithic-architecture>. Citado na página 20.

BáNáTI, A. et al. Authentication and authorization orchestrator for microservice-based software architectures. In: **41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)**. [S.l.: s.n.], 2018. p. 1180–1184. Citado 4 vezes nas páginas 23, 24, 35 e 36.

BARABANOV, A.; MAKRUSHIN, D. Authentication and authorization in microservice-based systems: Survey of architecture patterns. **Voprosy kiberbezopasnosti**, p. 32–43, 01 2020. Citado 2 vezes nas páginas 23 e 25.

BHUTADA, S.; JYOTHI, K. Enhancing security to the microservice (ms) architecture by implementing authentication and authorization (aa) service using docker and kubernetes. **International Journal of Innovative Technology and Exploring Engineering**, Blue Eyes Intelligence Engineering and Sciences Publication, v. 8, n. 6, p. 401–407, 2019. ISSN 22783075. Citado 3 vezes nas páginas 22, 35 e 36.

CHATTERJEE, A.; PRINZ, A. Applying spring security framework with keycloak-based oauth2 to protect microservice architecture apis: A case study. **Sensors**, v. 22, n. 5, 2022. ISSN 14248220. Citado 4 vezes nas páginas 22, 35, 36 e 37.

CHATZIGEORGIOU, A. A. S. B. P. A. M. V. A. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. **Applied Sciences**, v. 106, n. 30, 2019. Citado na página 41.

CHEN, Y.; HUANG, S.; KING, C. Protection of authentication credentials of cloud services. 2019. Disponível em: <https://patentimages.storage.googleapis.com/fa/39/aa/38cc0217793f9c/US10454921.pdf>. Citado na página 22.

CLOUD, A. **Implement access control for a Dubbo application by using service authentication**. 2023. Disponível em: <https://www.alibabacloud.com/help/en/edas/user-guide/implement-access-control-of-dubbo-applications-through-service-authentication>. Citado na página 36.

DAS, D. et al. On automated rbac assessment by constructing a centralized perspective for microservice mesh. **PeerJ Computer Science**, v. 7, p. 1–24, 2021. ISSN 23765992. Citado 2 vezes nas páginas 35 e 36.

DYBa, T.; DINGSØYR, T. Strength of evidence in systematic reviews in software engineering. In: **Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement**. New York, NY, USA: Association for Computing Machinery, 2008. (ESEM '08), p. 178—-187. ISBN 9781595939715. Citado na página 32.

EDGE, G. **What is Gloo Edge**. 2023. Disponível em: <https://docs.solo.io/gloo-edge/latest/>. Citado na página 36.

FENGXUAN, W. et al. Research on service security reinforcement scheme based on application gateway. In: **2023 IEEE 3rd International Conference on Data Science and Computer Application (ICDSCA)**. [S.l.: s.n.], 2023. p. 113–116. Citado na página 25.

FOWLER, M.; PARSONS, R. **Domain-specific languages**. Upper Saddle River, NJ: Addison-Wesley, 2011. ISBN 0321712943 9780321712943. Disponível em: <http://www.amazon.co.uk/Domain-Specific-Languages-Addison-Wesley-Signature/dp/0321712943/ref=wl_it_dp_o_pC_nS_nC?ie=UTF8&colid=18QO9YU77BZG4&coliid=I8JA9JG7ADISE>. Citado na página 50.

GATEWAY, E. **Credential Management**. 2023. Disponível em: <https://www.express-gateway.io/docs/credential-management/>. Citado na página 36.

HANNOUSSE, A.; YAHIOUCHE, S. Securing microservices and microservice architectures: A systematic mapping study. **Computer Science Review**, v. 41, 2021. ISSN 15740137. Citado 4 vezes nas páginas 21, 22, 23 e 44.

HE, X.; YANG, X. Authentication and authorization of end user in microservice architecture. In: . Guilin, China: [s.n.], 2017. v. 910, n. 1. ISSN 17426588. Authentication and authorization;Client sides;Distributed systems;End users;Fine granularity;Inherent complexity;Low coupling;Monolithic architecture;. Citado 5 vezes nas páginas 20, 22, 23, 35 e 36.

HOSTMIDIA. **Logs do sistema operacional – o que são e para que servem?** 2024. Disponível em: <https://www.hostmidia.com.br/blog/logs-do-sistema-operacional/#:~:text=Logs%20s%C3%A3o%20arquivos%20de%20texto,algum%20momento%20podem%20ser%20%C3%BAteis.> Citado na página 25.

KHAN, A. R. Access control in cloud computing environment. **ARPN Journal of Engineering and Applied Sciences**, v. 7, p. 613–615, 05 2012. Citado na página 22.

KITCHENHAM, B.; BRERETON, P. A systematic review of systematic review process research in software engineering. **Inf. Softw. Technol.**, Butterworth-Heinemann, USA, v. 55, n. 12, p. 2049—-2075, dec 2013. ISSN 0950–5849. Citado 2 vezes nas páginas 31 e 42.

KONG. **Kong**. 2023. Disponível em: <https://github.com/Kong/kong>. Citado na página 37.

KRAFZIG, D.; BANKE, K.; SLAMA, D. **Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)**. USA: Prentice Hall PTR, 2004. ISBN 0131465759. Citado na página 20.

KRAKEND. **KrakenD Community Edition Documentação**. 2023. Disponível em: <https://www.krakend.io/docs/overview/>. Citado na página 37.

LEE, W.-T.; TSAI, M.-K. Implementing a php api gateway based on microservices architecture. In: **48th Annual Computers, Software, and Applications Conference**. [S.l.]: IEEE, 2024. p. 1578–1579. Citado 2 vezes nas páginas 26 e 27.

LIU, H. et al. Building a private cloud based on microservices for computer science laboratory in universities. In: Y., L. S. D. Y. M. J. C. (Ed.). **7th International Conference on Information Science and Control Engineering, ICISCE 2020**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2020. p. 379–384. ISBN 9781728164069. Citado 3 vezes nas páginas 25, 35 e 36.

LIU, W.; SCHMIEDEHAUSEN, H.; WANG, Z. Systems and methods for providing authentication in a microservice system. 2024. Disponível em: <https://patentimages.storage.googleapis.com/45/43/a5/5a39268aa102c0/US11863547.pdf>. Citado na página 23.

LUZ, W. et al. An experience report on the adoption of microservices in three brazilian government institutions. In: **Proceedings of the XXXII Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2018. (SBES '18), p. 32–41. ISBN 9781450365031. Disponível em: <https://doi.org/10.1145/3266237.3266262>. Citado 2 vezes nas páginas 16 e 20.

MAJUMDER, A.; NAMASUDRA, S.; NATH, S. **Taxonomy and Classification of Access Control Models for Cloud Environments**. [S.l.: s.n.], 2014. 23-53 p. ISBN 978-1-4471-6451-7. Citado na página 22.

MARCONI, M. A.; LAKATOS, E. M. **Fundamentos de Metodologia Científica**. 8. ed. [S.l.]: Atlas, 2017. ISBN 9788597010763. Citado na página 18.

MCLARTY, M.; WILSON, R.; MORRISON, S. Securing microservices apis. In: . MA, USA: O'Reilly, 2018. Citado na página 23.

MEIER, J. **Performance Testing Guidance for Web Applications: Patterns & Practices**. Microsoft Press, 2007. (ITPro collection). ISBN 9780735625709. Disponível em: <https://books.google.com.br/books?id=a_yiPQAACAAJ>. Citado na página 64.

MELTON, R. Securing a Cloud-Native C2 Architecture Using SSO and JWT. In: **2021 IEEE Aerospace Conference (50100)**. [S.l.: s.n.], 2021. p. 1–8. ISSN 1095-323X. Citado 5 vezes nas páginas 22, 23, 35, 36 e 37.

MONTESI, F.; WEBER, J. Circuit breakers, discovery, and api gateways in microservices. 09 2016. Citado na página 25.

MOREIRA, P.; RIBEIRO, A.; SILVA, J. M. AGE: Automatic Performance Evaluation of API Gateways. In: **Symposium on Computers and Communications**. [S.l.]: IEEE, 2023. p. 405–410. Citado 2 vezes nas páginas 26 e 27.

OKTARIA, D. et al. Design of api gateway as middleware on platform as a service. **Indonesia Journal on Computing (Indo-JC)**, v. 6, n. 3, p. 47–62, Dec. 2021. Disponível em: <https://socj.telkomuniversity.ac.id/ojs/index.php/indojc/article/view/597>. Citado 2 vezes nas páginas 26 e 27.

PALLISTER, T. **Big Picture**. 2023. Disponível em: <https://ocelot.readthedocs.io/en/latest/introduction/bigpicture.html>. Citado na página 37.

PASOMSUP, C.; LIMPIYAKORN, Y. HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager. In: **International Conference on Big Data Engineering and Education**. [S.l.: s.n.], 2021. (BDEE'21), p. 177–181. Citado 4 vezes nas páginas 20, 35, 36 e 39.

PEFFERS, K. et al. A design science research methodology for information systems research. **Journal of Management Information Systems**, v. 24, p. 45–77, 01 2007. Citado na página 18.

PEREIRA-VALE, A. et al. Security in microservice-based systems: A multivocal literature review. **Computers Security**, v. 103, p. 25, 01 2021. Citado na página 22.

PEREIRA-VALE, A. et al. Security Mechanisms Used in Microservices-Based Systems: A Systematic Mapping. In: **XLV Latin American Computing Conference (CLEI)**. Panama, Panama: IEEE, 2019. p. 1–10. Citado 2 vezes nas páginas 21 e 43.

PONCE, F. et al. Smells and refactorings for microservices security: A multivocal literature review. **Journal of Systems and Software**, Elsevier Inc., v. 192, 2022. ISSN 01641212. Citado na página 43.

PONTAROLLI, R. P. et al. Towards Security Mechanisms for an Industrial Microservice-Oriented Architecture. In: **2021 14th IEEE International Conference on Industry Applications (INDUSCON)**. [S.l.: s.n.], 2021. p. 679–685. Citado 3 vezes nas páginas 35, 36 e 39.

PREUVENEERS, D.; JOOSEN, W. Towards Multi-party Policy-based Access Control in Federations of Cloud and Edge Microservices. In: **IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)**. [S.l.: s.n.], 2019. p. 29–38. Citado 3 vezes nas páginas 25, 35 e 36.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição**. [S.l.]: Editora Feevale, 2013. Citado na página 18.

RAJ, P.; VANGA, S.; CHAUDHARY, A. **Microservices Security: The Concerns and the Solution Approaches**. [S.l.]: Wiley-IEEE Press, 2023. 289–298 p. ISBN 9781119814771. Citado 5 vezes nas páginas 16, 24, 30, 35 e 36.

RANAWAKA, I. et al. Custos: Security Middleware for Science Gateways. In: **Practice and Experience in Advanced Research Computing**. New York, NY, USA: ACM, 2020. (PEARC'20), p. 278—-284. ISBN 9781450366892. Citado 3 vezes nas páginas 35, 36 e 39.

ROQUES, A. **PlantUML**. 2024. Disponível em: <https://plantuml.com>. Citado na página 50.

RUNESON, P.; ENGSTRÖM, E.; STOREY, M.-A. **The Design Science Paradigm as a Frame for Empirical Software Engineering**. Germany: Springer, 2020. 127–147 p. ISBN 978-3-030-32488-9. Citado na página 18.

SANCHEZ, I. H.; CARNELL, J. **Spring Microservices in Action, Second Edition**. [S.l.]: IEEE, 2021. Citado 2 vezes nas páginas 37 e 52.

SHULIN, Y.; JIEPING, H. Research on Unified Authentication and Authorization in Microservice Architecture. In: **IEEE 20th International Conference on Communication Technology (ICCT)**. [S.l.]: IEEE, 2020. p. 1169–1173. ISSN 2576-7828. Citado 3 vezes nas páginas 22, 35 e 36.

SINGH, A.; RAJ, V.; SADAM, R. **Integration of Attribute-Based Access Control in Microservices Architecture**. [S.l.: s.n.], 2022. 681-690 p. ISBN 978-981-16-5986-7. Citado na página 23.

SNGER, N.; ABECK, S. Authentication and authorization in microservice-based applications. In: . Hamburg, Germany: [s.n.], 2022. P–326, p. 207–218. ISSN 16175468. Citado 4 vezes nas páginas 16, 35, 36 e 37.

SORDI, J. O. D.; MARINHO, B. d. L.; NAGY, M. Benefícios da arquitetura de software orientada a serviços para as empresas: análise da experiência do abn amro brasil. **JISTEM : Journal of Information Systems and Technology Management**, 2006. Citado na página 20.

TECHNOLOGIES, T. **Tyk Gateway Open Source (OSS)**. 2023. Disponível em: <https://tyk.io/docs/tyk-oss-gateway/>. Citado na página 37.

TECHNOLOGY, G. Y. I. **Goku API Gateway**. 2023. Disponível em: <https://www.gokuapi.com/>. Citado na página 36.

TOMIć, M. et al. Towards applying api gateway to support microservice architectures for embedded systems. In: . [S.l.: s.n.], 2022. Citado 2 vezes nas páginas 26 e 27.

TRAVASSOS, G. H.; BARROS, M. O. Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. In: **2nd Workshop on empirical software engineering the future of empirical studies in software engineering**. [S.l.: s.n.], 2003. p. 117–130. Citado na página 59.

TRIARTONO, Z.; NEGARA, R. M.; SUSSI. Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorization System. In: **6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)**. [S.l.: s.n.], 2019. (EECSI'20), p. 259–263. Citado 3 vezes nas páginas 29, 35 e 36.

UFSM/CPD. **SIE**. 2024. Disponível em: <https://www.ufsm.br/orgaos-suplementares/cpd/servicos/sie>. Citado na página 15.

VENABLE, J.; BASKERVILLE, R. Eating our own cooking: Toward a more rigorous design science of research methods. **Electronic Journal of Business Research Methods**, v. 10, p. 141–153, 01 2012. Citado na página 18.

VáZQUEZ-INGELMO, A.; GARCíA-HOLGADO, A.; GARCíA-PEñALVO, F. J. C4 model in a software engineering subject to ease the comprehension of uml and the software. In: **2020 IEEE Global Engineering Education Conference (EDUCON)**. [S.l.: s.n.], 2020. p. 919–924. Citado na página 50.

WOHLIN, C. et al. Guidelines for snowballing in systematic literature studies and a replication in software engineering. **Empirical software engineering**, Springer, v. 21, n. 3, p. 797–829, 2016. Citado 2 vezes nas páginas 43 e 44.

WOHLIN, C. et al. **Experimentation in Software Engineering**. [S.l.]: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434. Citado 3 vezes nas páginas 42, 59 e 84.

XIONG, Q.; LI, W. Design and Implementation of Microservices Gateway Based on Spring Cloud Zuul. In: **CIBDA 2022; 3rd International Conference on Computer Information and Big Data Applications**. [S.l.: s.n.], 2022. p. 1–5. Citado 4 vezes nas páginas 20, 35, 36 e 37.

XU, R.; JIN, W.; KIM, D. Microservice security agent based on api gateway in edge computing. **Sensors (Switzerland)**, v. 19, n. 22, 2019. ISSN 14248220. Citado 6 vezes nas páginas 16, 25, 35, 36, 37 e 39.

XU, S. et al. Exploring Folksonomy for Personalized Search. In: **31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval**. New York, NY, USA: ACM, 2008. (SIGIR'08), p. 155–162. ISBN 9781605581644. Citado 2 vezes nas páginas 40 e 41.

YANG, J. et al. User Fast Authentication Method Based on Microservices. In: **IEEE International Conference on Power Electronics, Computer Applications (ICPECA)**. [S.l.: s.n.], 2021. p. 93–98. Citado 7 vezes nas páginas 21, 24, 35, 36, 37, 38 e 39.

YARYGINA, T.; BAGGE, A. Overcoming security challenges in microservice architectures. In: **12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018**. [S.l.]: IEEE, 2018. p. 11–20. ISBN 9781538652060. Citado 4 vezes nas páginas 20, 35, 36 e 39.

YU, D. et al. A survey on security issues in services communication of microservices-enabled fog applications. **Concurrency and Computation: Practice and Experience**, v. 31, n. 22, p. e4436, 2019. E4436 cpe.4436. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4436>. Citado 2 vezes nas páginas 23 e 38.

ZHAO, J.; SUN, J. Research on access control model based on rbac model in microservice environment. **Journal of Physics: Conference Series**, IOP Publishing, v. 1437, n. 1, p. 012031, jan 2020. Disponível em: <https://dx.doi.org/10.1088/1742-6596/1437/1/012031>. Citado 2 vezes nas páginas 22 e 25.

ZUO, X. et al. An api gateway design strategy optimized for persistence and coupling. **Advances in Engineering Software**, v. 148, p. 102878, 2020. ISSN 0965-9978. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0965997820304452>. Citado 2 vezes nas páginas 26 e 27.