# Serviço para *gerenciamento de identidades e acessos* baseado em *Arquitetura Limpa e Microsserviços*

**Trabalho de Conclusão do Curso Superior de**

**Tecnologia em Sistemas Para Internet**

**Giovani de Souza Boff**

**Orientador: Rodrigo Prestes Machado**

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul
(IFRS), Av. Cel. Vicente, 281, Porto Alegre – RS – Brasil
giovani_souzaboff@hotmail.com, rodrigo.prestes@poa.ifrs.edu.br

**Resumo**: A arquitetura de microsserviços, amplamente adotada por grandes empresas de tecnologia, possibilita a escalabilidade eficiente do software. Complementada pela Arquitetura Limpa, cria aplicações robustas e extensíveis. No entanto, a implementação desse tipo de sistema apresenta desafios, como a decomposição adequada do sistema monolítico em microsserviços, exigindo uma análise cuidadosa para definir limites de contexto e interfaces claras entre os serviços. A escalabilidade individual de cada serviço também é crucial para lidar com variações de demanda. Diante desses desafios, o objetivo deste trabalho foi desenvolver uma ferramenta que simplifica as principais funcionalidades relacionadas ao controle de acessos e identidades em aplicações baseadas em arquitetura limpa e microsserviços. Essa ferramenta visa disponibilizar funcionalidades essenciais encontradas em serviços desse tipo.Para alcançar esse objetivo, o serviço proposto passou por testes abrangentes em seus componentes. Foram realizados testes tanto manuais quanto automatizados para verificar o comportamento da aplicação em cenários reais. Esses testes garantiram a segurança e a efetividade do sistema, comprovando que foi possível criar um microserviço capaz de disponibilizar funcionalidades essenciais para o gerenciamento de identidades e acessos.

**Palavras-chave:** Arquitetura de Software, Microsserviços, Arquitetura Limpa, Gestão de Identidade e acessos.

## 1.INTRODUÇÃO

A estrutura de controle de acessos e identidades desempenha um papel fundamental na segurança dos dados em aplicações. No entanto, soluções comerciais existentes, como o

KEYCLOAK (RED HAT, 2021) e o GLUU SERVER (GLUU INC, 2021), podem ser complexas e apresentar funcionalidades desnecessárias, o que dificulta sua implantação e torna o uso dispendioso e pouco escalável.

Diante desse cenário, a arquitetura de microsserviços e a abordagem da arquitetura limpa surgem como alternativas promissoras. Embora os conceitos fundamentais por trás dessa arquitetura não sejam novos, a aplicação de microsserviços é atual e sua adoção tem sido motivada, em parte, pelos desafios de escalabilidade, alto consumo de recursos, alto custo de manutenção e implementação de novas funcionalidades e dificuldades em adotar novas tecnologias que surgem quando sistemas complexos de software estão contidos e são implantados como uma grande aplicação monolítica (FOWLER, 2017). Em complemento a isso, a arquitetura limpa, apresentada por Martin (2019), define um padrão de arquitetura, propondo o encapsulamento da lógica de negócio da aplicação, mantendo o sistema desenvolvido agnóstico a frameworks e de fácil manutenção. Em conjunto com a arquitetura de microsserviços, essa abordagem permite uma aplicação mais compacta e escalável.

Com isso, constatou-se que as principais ferramentas possuem uma abordagem complexa e é direcionada para aplicações mais robustas. Diante disso, identificou-se a necessidade de desenvolver um serviço mais simples, com o objetivo de oferecer funcionalidades mais essenciais. Além disso, esse trabalho busca proporcionar um modelo básico para a implementação de autenticação e autorização em sistemas que demandam essas funcionalidades. A proposta é desenvolver um microsserviço com uma arquitetura limpa de fácil manutenção e alta extensibilidade, que possa ser mantido e evoluído pela comunidade open source futuramente.

## 2. FUNDAMENTAÇÃO TEÓRICA

### 2.1. GERENCIAMENTO DE IDENTIDADES E ACESSOS

Um sistema de gerenciamento de identidades e acesso (*Identity and Access Management - IAM*) é uma solução de software que gerencia o acesso de usuários a recursos de uma organização, tais como aplicativos, dados e sistemas. O IAM envolve diversos processos, como autenticação, autorização, gerenciamento de identidades e provisionamento de usuários. De acordo com Rhand (2017), as normas da família ISO/IEC 27000 (ISO/IEC 27000 et al., 2018) definem os seguintes conceitos fundamentais:

Primeiro a identificação que refere-se aos métodos utilizados para fornecer a um sujeito

(uma entidade que solicita acessos) uma identidade reconhecível. Isso pode ser realizado por meio de uma conta de usuário, passaporte ou qualquer outra forma de identificação única. Por segundo a autenticação que abrange os métodos empregados para garantir que um sujeito seja realmente quem afirma ser. Diversas técnicas podem ser utilizadas nesse processo, como senhas, tokens de segurança, impressões digitais, entre outras, a fim de verificar a identidade do indivíduo. E por fim a autorização que diz respeito aos métodos que controlam quais ações um sujeito pode realizar em um objeto. Isso envolve a definição de permissões e restrições, tanto para o sujeito quanto para o objeto em questão. Por exemplo, são estabelecidas listas de permissões para determinar quais ações podem ser executadas pelo sujeito em relação a um objeto específico.

Esses conceitos fundamentais são essenciais para estabelecer um sistema eficiente de controle de acesso. A identificação garante que cada sujeito possua uma identidade única e reconhecível, a autenticação valida essa identidade e a autorização controla as ações permitidas para cada sujeito em relação a um objeto. Ao compreender e aplicar adequadamente esses conceitos, é possível desenvolver um sistema de gerenciamento de identidades e acesso robusto e seguro.

## 2.2.ARQUITETURA DE SOFTWARE

A arquitetura de *software*, de uma maneira geral, define a forma dada a um sistema pelo seu desenvolvedor, tal forma está na divisão dos componentes, na organização desses componentes e nos modos que essas estruturas dos sistemas se comunicam entre si (MARTIN, 2019).

A escolha de uma arquitetura bem definida, influencia diretamente na performance, qualidade, escalabilidade e na facilidade de manutenção de um sistema, tendo um impacto no sucesso do desenvolvimento do sistema, segundo Sommerville (2011): “Ao tomar decisões sobre a arquitetura de um sistema, você deve conhecer os padrões comuns, bem como saber onde eles podem ser usados e quais são seus pontos fortes e fracos.”.

Há diversos princípios de padrões utilizados nos sistemas, não se limitando a um único estilo de arquitetura, o grande desafio de um software bem estruturado é capacidade de manter de forma planejada e coesa o foco em sua arquitetura.

Pode-se compreender melhor a partir do propósito de uma arquitetura, definido por Martin (2019): “As boas arquiteturas devem ser centradas em casos de uso para que os arquitetos

possam descrever com segurança as estruturas que suportam esses casos de uso, sem se comprometer com *frameworks*, ferramentas e ambientes".

## 2.3. ARQUITETURA DE MICROSSERVIÇOS

A arquitetura baseada em microsserviços é um modelo arquitetônico de desenvolvimento de software que baseia-se em separar serviços em pequenos domínios responsáveis por operações únicas.

O objetivo dessa arquitetura é construir um conjunto de pequenas aplicações, cada uma responsável por executar uma função, é permitir que cada microsserviço seja autônomo, independente e autossuficiente. (FOWLER, 2017).

A partir da implementação de microsserviços, uma aplicação pode ser facilmente escalada tanto horizontalmente quanto verticalmente, a produtividade e a velocidade do desenvolvedor aumentam dramaticamente. (FOWLER, 2017).

## 2.4. ARQUITETURA DE SOFTWARE LIMPA

A arquitetura de software limpa tem como objetivo organizar o código desenvolvido de forma que seu funcionamento seja conciso e siga um fluxo bem definido. Essa arquitetura possui como base a divisão dos elementos do sistema em camadas de modo que esses elementos se comuniquem seguindo um fluxo de fora para dentro, ou seja, as camadas mais internas não possuem conhecimento das mais externas, propondo uma organização do código de forma que encapsule a lógica de negócios, independente de frameworks ou bibliotecas externas.

Dantas; Casillo; Neto (2021) apresentam em sua proposta de projeto uma visão sobre as camadas que definem a arquitetura limpa de Martin (2019) e como são separadas, explicando-as de forma objetiva. As camadas de desenvolvimento propostas pela arquitetura limpa são separadas da seguinte maneira (APÊNDICE 1):

As entidades são classes que representam conceitos centrais do domínio da aplicação e incorporam regras de negócio comumente utilizadas. Por sua vez, os casos de uso são classes mais específicas, responsáveis por lidar com funcionalidades do sistema, como o cadastro de novos usuários, interagindo com as entidades para validar os dados inseridos. Os adaptadores/controles têm a tarefa de converter e adaptar os dados, preparando-os para serem processados pela camada de destino. Já na camada de frameworks/dispositivos externos, encontram-se as implementações provenientes de bibliotecas e frameworks externos. Essa

camada pode oferecer recursos de persistência, interfaces e outros serviços, como o envio de e-mails Dantas; Casillo; Neto (2021). Essa estrutura em camadas proporciona uma organização eficiente e modular do sistema, facilitando seu desenvolvimento e manutenção.

## 3.TRABALHOS RELACIONADOS

Foi conduzida uma pesquisa com o intuito de investigar sistemas e estudos relacionados ao desenvolvimento de microsserviços para gerenciamento de acessos e identidades, considerando diversas propostas de implementação para este gerenciamento.

Na seção a seguir, destaca-se projetos encontrados e que se relacionam ao tema do trabalho proposto, onde são feitas as comparações das suas funcionalidades

### 3.1.KEYCLOAK

O Keycloak é um produto de software de código aberto focado no gerenciamento de identidade e acesso para aplicações e serviços. É licenciado pela Apache e mantido pela Red Hat. (RED HAT, 2021)

O Software oferece diversas funcionalidades, entre os recursos oferecidos, destacam-se a criação de usuários, permitindo o registro e inclusão de novos membros no sistema, o sistema de login, garantindo a autenticação e acesso seguro aos usuários autorizados, a integração com o Active Directory, facilitando a sincronização e gerenciamento centralizado das informações de usuários, a ativação de usuários por confirmação de e-mail, possibilitando a verificação e validação das contas por meio do envio de um e-mail de confirmação, e a criação de grupos de usuários, permitindo a organização e atribuição de permissões específicas para conjuntos de usuários.

Dentre suas vantagens, é um projeto mantido pela comunidade, além de trazer a solução para autenticação de autorização. Porém possui uma documentação extensa, por oferecer muitas opções para a implementação dele, o que acaba levando um bom tempo para o entendimento das suas funções.

O Keycloak é focado em um mercado onde aplicações precisam dessa robustez oferecidas por ele, porém não sendo utilizado em sua totalidade, por oferecer muitas funcionalidades desnecessárias conforme sua integração.

### 3.2.GLUU SERVER

O Gluu Server é um dos produtos oferecidos pela Gluu Inc, possui foco em gerenciamento

de identidade e acessos, assim como o Keycloak. O seu grande diferencial é oferecer além de autenticação e autorização, trazendo diversos outros serviços que acoplam outras funcionalidades no qual são mais detalhadas em sua documentação. (GLUU INC, 2021)

A principal ideia do Gluu Server é oferecer ao mercado, funcionalidades de segurança da informação de maneira desacoplada, separando-o em diferentes produtos.

## 3.3.SERVIÇO PROPOSTO

O microsserviço desenvolvido por esse projeto, busca oferecer uma fácil manutenção de seu código, aplicado a sua estrutura arquitetural, onde o foco principal é implementar as funcionalidades mais comuns encontradas nas ferramentas apresentadas anteriormente, em conjunto a arquitetura limpa que facilite a implementação de novas funcionalidades e a arquitetura de microsserviços.

Além das funcionalidades comuns encontradas nas ferramentas mencionadas, o microsserviço desenvolvido também inclui recursos de autenticação de dois fatores e autenticação WebAuthn. A autenticação de dois fatores acrescenta uma camada extra de segurança ao exigir que os usuários forneçam um segundo fator de autenticação, como um código enviado por mensagem de texto ou um aplicativo de autenticação. Isso reduz o risco de acesso não autorizado, mesmo que a senha seja comprometida. A autenticação WebAuthn, por sua vez, permite que os usuários façam login de forma segura em aplicativos e sites, eliminando a necessidade de senhas e utilizando criptografia assimétrica e chaves públicas e privadas. Esses recursos reforçam a segurança das contas dos usuários, protegendo contra invasões e roubo de identidade, ao mesmo tempo em que oferecem uma experiência de autenticação mais conveniente e confiável.

Ao combinar a arquitetura limpa com a arquitetura de microsserviços, projetou-se um serviço que seja facilmente mantido, permitindo a adição de novas funcionalidades de forma ágil e escalável. Essa abordagem também facilitará a integração com outras tecnologias e a colaboração da comunidade open source, proporcionando um ecossistema de desenvolvimento mais dinâmico e colaborativo.

**Tabela 1 - Comparação de Sistemas**

| Projeto | Serviço Proposto | Keycloak | Gluu Server |
|---|---|---|---|
| Open Source | Sim | Sim | Sim |
| Criação e gerenciamento de Usuários | Sim | Sim | Sim |
| Autenticação JWT | Sim | Sim | Sim |
| Autenticação WebAuthn | Sim | Não | Não |
| Autenticação de Dois Fatores | Sim | Não | Não |
| Estrutura de integração simples | Sim | Não | Não |

Fonte: elaborada pelo autor.

## 4.METODOLOGIA DE PESQUISA

Este trabalho utiliza a metodologia de *Design Science Research Process* (PEFFERS et *al.*, 2020). Esse método consiste em 6 etapas para validação de criações de tecnologias e trabalhos voltados para a área de sistemas de informação e foi selecionado, pois corresponde com o desenvolvimento e verificação do sistema proposto neste artigo.

### 4.1.ETAPAS DA METODOLOGIA

O *Design Science Research Process* (DSRP), é dividido em 6 etapas: identificação de problema e motivação, objetivos da solução, design e desenvolvimento, demonstração, validação e comunicação. As etapas são descritas a seguir:

#### 4.1.1.Identificação do problema e motivação

Durante a pesquisa realizada para este artigo, foram identificadas ferramentas que implementam as funcionalidades oferecidas pelo sistema proposto, porém a maioria oferece de

maneira complexa, tendo foco em um mercado para aplicações mais robustas. Com isso, foi observado a necessidade de implementar um sistema mais simples, com foco de disponibilizar um conjunto de funcionalidades essenciais, além de possibilitar o estudo em cima de sua estrutura focada em sua extensibilidade e na comunidade *open source*.

### 4.1.2. Objetivos da solução

Servir como modelo básico na implementação de autenticação e autorização em sistemas que necessitam dessas funcionalidades, buscando implementar uma arquitetura que possa ser facilmente mantido pela comunidade *open source* e extensível, tendo o seu código fonte hospedado no site GitHub, e fazendo parte dos serviços oferecidos pela plataforma Orion Service.

### 4.1.3. Design e desenvolvimento

O sistema adota uma combinação da arquitetura de microsserviços com a arquitetura de software limpa, proporcionando uma estrutura robusta e modular. Utiliza o framework Quarkus (QUARKUS, [s. d.]) e banco de dados relacional para armazenamento dos dados dos usuários. Os testes são realizados com o framework JUnit. Essas escolhas tecnológicas permitem um ambiente ágil e escalável, com a implementação de funcionalidades essenciais de gerenciamento de identidades e acessos. A colaboração com a comunidade é facilitada devido ao uso de soluções de código aberto.

O desenvolvimento segue a metodologia ágil (MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT, 2001), com entregas separadas por sprints e prazos definidos. A plataforma Orion Service é o cliente final do projeto.

### 4.1.4. Demonstração

A demonstração do serviço proposto foi feita por meio de testes de unidade, de integração e manuais, baseados em cenários de testes que descrevem o comportamento esperado pelo microserviço desenvolvido por este trabalho.

### 4.1.5. Validação

Foi utilizado o framework JUnit, uma ferramenta de código aberto amplamente adotada para realizar testes automatizados na linguagem Java. Essa escolha permite a criação e execução, incluindo testes de unidade e testes de integração. Os testes de unidade analisaram o funcionamento das partes individuais do serviço, enquanto os testes de integração avaliaram a

interação entre os componentes do sistema.

#### 4.1.6.Comunicação

A comunicação é feita através da apresentação do artigo e sistema diante da banca avaliadora do curso de Tecnologia em Sistemas Para Internet do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS) - Campus Porto Alegre e disponibilização do código fonte do projeto na plataforma GitHub[1].

## 5.DESCRIÇÃO DO SISTEMA

O microsserviço proposto por este trabalho foi desenvolvido com o objetivo de implementar as funcionalidades essenciais relacionadas ao gerenciamento de identidades e acessos. Essas funcionalidades são expostas a outros serviços por meio de rotas HTTP seguindo o design REST. O microsserviço permite a criação e gerenciamento de usuários, além de oferecer recursos de autenticação e autorização utilizando o padrão de criptografia de token JWT. A estrutura do código base segue a arquitetura limpa proposta por Martin (2019), porém adaptada para atender às necessidades específicas do microsserviço em questão. Essa abordagem busca garantir a modularização e organização das regras de negócio, separando-as das camadas externas e tornando o código mais coeso e fácil de manter.

O microsserviço oferece aos clientes a capacidade de autenticar-se por meio de tokens JWT, garantindo uma forma segura de autenticação para acessar as funcionalidades disponíveis. Os clientes podem criar novos usuários, remover usuários existentes e recuperar senhas, fornecendo seu e-mail registrado. Além disso, eles têm a opção de atualizar suas senhas regularmente para manter suas contas seguras.O microsserviço também fornece recursos de validação de e-mail e autenticação de dois fatores para reforçar a segurança das contas dos usuários. A validação de e-mail é realizada através do envio de um e-mail de validação após a criação da conta, garantindo a autenticidade do endereço fornecido pelo cliente. A autenticação de dois fatores é uma opção adicional para aumentar a segurança das contas dos clientes. Ao ativá-la, o cliente deve fornecer um segundo fator de autenticação, como um código gerado por um aplicativo de autenticação. Isso reduz o risco de acesso não autorizado às contas.

Em resumo, o microsserviço oferece um conjunto de funcionalidades essenciais para autenticação e segurança das contas de usuários, permitindo que os clientes acessem o sistema de forma segura através do uso de tokens JWT

[1] Disponível em: https://github.com/orion-services/users Acesso em: 05 jul. 2023.

## 5.1.ARQUITETURA LIMPA DO SERVIÇO

O sistema proposto adota uma arquitetura de código que combina os princípios da Arquitetura de Software Limpa de Martin (2019). Essa arquitetura tem como objetivo garantir um fluxo eficiente de gerenciamento de acesso e identidades, dividindo o sistema em camadas independentes (APÊNDICE 2A).

A primeira camada é a Camada de Domínio (Domain Layer), que representa o núcleo do sistema. Nessa camada, estão concentradas as regras de negócio relacionadas à entidade principal da aplicação, que no caso é o "User" (usuário) e também é utilizada como entidade do banco de dados. Essa camada é a mais interna da arquitetura e é responsável por encapsular as regras de negócio de forma independente das outras camadas, permitindo que sejam desenvolvidas e testadas de forma isolada. A segunda camada é a Camada de Uso de Casos (Data Layer), responsável por implementar os casos de uso definidos pela aplicação. Nessa camada, são aplicadas as regras mais relacionadas à aplicação em si. Aqui, os casos de uso do domínio são conectados ao banco de dados. Além disso, essa camada trata as respostas e erros gerados pela aplicação, garantindo uma lógica coesa e consistente. A Camada de Infraestrutura (Infra Layer) agrupa as implementações de serviços externos utilizados pela aplicação. Nessa camada, estão incluídas as integrações com o banco de dados e frameworks que auxiliam no desenvolvimento. Ela permite a comunicação com recursos externos e o gerenciamento de aspectos técnicos do sistema. Por fim, temos a Camada de Apresentação (Presentation Layer), que é a camada mais externa da arquitetura. Nessa camada, são agrupados os controles de navegação do microsserviço . Ela fornece as rotas de comunicação com as camadas mais internas, permitindo a interação do sistema com os usuários ou outros sistemas externos. Por exemplo, é nessa camada que são definidas as rotas HTTP para clientes externos, como a rota de criação de usuários (APÊNDICE 2B).

Essa estrutura arquitetural baseada em camadas independentes proporciona uma organização clara e modular do sistema. Cada camada possui responsabilidades específicas, facilitando o desenvolvimento, a manutenção e a evolução do sistema. Além disso, essa abordagem permitiu que as camadas sejam testadas de forma isolada, promovendo a qualidade e a escalabilidade da aplicação.

## 5.2.Casos de uso

Nesta seção, são abordados os casos de uso que abrangem o sistema proposto por este trabalho (APÊNDICE 3).

Dentre os requisitos apresentados, o Cliente tem acesso a várias funcionalidades relacionadas à autenticação e gerenciamento de usuários.

A funcionalidade de "Autenticação" permite que o Cliente obtenha um token de autorização e autenticação por meio de uma rota específica. Esse token é essencial para acessar outras funcionalidades do sistema relacionadas ao gerenciamento de usuários. Uma vez obtido o token de autorização e autenticação, o Cliente pode desfrutar de várias funcionalidades. Por exemplo, a funcionalidade "Criar Usuário" possibilita adicionar novos usuários ao sistema, fornecendo as informações necessárias, como nome, e-mail e senha. Além disso, o Cliente pode utilizar a funcionalidade "Deletar Usuário" para remover um usuário existente do sistema. Ao realizar uma requisição para a rota de exclusão de usuário e fornecer o ID do usuário a ser excluído, o sistema verifica as permissões do Cliente com base no token de autorização e autenticação, localiza o usuário com base no ID fornecido e o exclui do sistema.

No caso de esquecimento da senha, o Cliente pode utilizar a funcionalidade "Recuperar Senha" para redefini-la. Ao fornecer seu e-mail registrado por meio da rota de recuperação de senha, o sistema verifica se o e-mail está associado a uma conta existente. Em caso afirmativo, o sistema gera um link de redefinição de senha e o envia para o e-mail do Cliente. Ao acessar o link, o Cliente pode fornecer uma nova senha, e o sistema atualiza a senha da conta com a nova informação.A funcionalidade "Atualizar Senha" permite ao Cliente modificar sua senha atual. Ao realizar uma requisição para a rota de atualização de senha e fornecer a senha atual e a nova senha desejada, o sistema valida a senha atual do Cliente e atualiza a senha da conta com a nova informação.Da mesma forma, o Cliente pode utilizar a funcionalidade "Atualizar Email" para modificar seu endereço de e-mail atual. Ao realizar uma requisição para a rota de atualização de e-mail, o Cliente fornece o novo endereço de e-mail desejado, e o sistema realiza a validação do novo endereço antes de atualizá-lo na conta do Cliente. Após a criação de uma conta no sistema, a funcionalidade "Validar E-mail" é acionada para confirmar a autenticidade do endereço de e-mail fornecido durante o processo de registro. O sistema envia um e-mail de validação para o endereço fornecido, e o Cliente precisa acessar sua caixa de entrada de e-mail, localizar a mensagem de validação enviada pelo sistema e seguir o link ou utilizar o código fornecido para validar o endereço de e-mail. Ao receber a solicitação de validação, o sistema verifica a validade do link ou código e atualiza o status de validação do endereço de e-mail na conta do Cliente.

Além disso, a funcionalidade "Autenticação de dois fatores" é responsável por fornecer uma camada adicional de segurança ao sistema. Ao ativar a autenticação de dois fatores nas configurações de segurança da conta, o Cliente pode utilizar códigos de validação gerados pelo sistema em conjunto com outros aplicativos de autenticação para validar seu acesso. Essa medida fortalece a proteção da conta do Cliente, exigindo a verificação em duas etapas. Por fim, o Cliente também tem a opção de utilizar a funcionalidade de "Autenticação com WebAuthn" para uma autenticação mais segura e sem senhas. Ao realizar uma requisição para a autenticação com WebAuthn, o Cliente se beneficia da autenticação baseada em criptografia assimétrica fornecida pelo WebAuthn. O sistema verifica e valida a autenticação do Cliente usando WebAuthn, proporcionando uma experiência de autenticação mais segura, eliminando a necessidade de senhas.

## 5.3.Casos de Testes

Nesta seção, são abordados os cenários de teste que visam avaliar se o sistema proposto neste trabalho alcançou seus objetivos.

Os casos de testes foram devidamente baseados nos casos de usos do microsserviço, levando em consideração o cliente como usuário do sistema.

**Tabela 2. Casos de testes**

| | **Cenário** | **Resultado esperado** |
|---|---|---|
| 1. | O usuário insere as credenciais corretas (nome de usuário e senha) e é autenticado com sucesso; | A aplicação deve retornar um token de autenticação válido. |
| 2. | O usuário insere as credenciais incorretas e recebe uma mensagem de erro informando que as credenciais estão inválidas; | A aplicação deve retornar uma mensagem de erro informando que as credenciais estão inválidas e não deve conceder acesso ao usuário. |

| | | |
|---|---|---|
| 3. | O usuário deixa campos em branco e recebe uma mensagem de erro informando que os campos são obrigatórios; | A aplicação deve retornar uma mensagem de erro informando que os campos são obrigatórios e não deve conceder acesso ao usuário. |
| 4. | O cliente cria um novo usuário com sucesso; | O novo usuário deve ser adicionado com sucesso à base de dados e as informações do usuário devem ser as mesmas que foram inseridas pelo usuário autenticado. |
| 5. | O cliente exclui um usuário com sucesso; | O usuário excluído não deve mais aparecer na lista de usuários. |
| 6. | O usuário insere o e-mail correto para recuperação de senha e recebe um e-mail com um link de recuperação de senha; | A aplicação deve enviar um e-mail com um link de recuperação de senha válido para o endereço de e-mail informado pelo usuário. |
| 7. | O usuário atualiza sua senha com sucesso | A senha do usuário deve ser atualizada corretamente na base de dados. |
| 8. | O usuário autenticado atualiza seu e-mail com sucesso; | O e-mail do usuário deve ser atualizado corretamente na base de dados. |
| 10. | O usuário ativa a autenticação de dois fatores e é capaz de gerar códigos de validação. | O código de validação deve ser gerado corretamente e a autenticação de dois fatores deve ser efetivada. |

| | | |
|---|---|---|
| | O usuário insere o código de validação gerado com sucesso e é autenticado corretamente. | |
| 11. | Dado um usuário registrado no sistema com uma chave WebAuthn associada<br><br>Quando o usuário tenta fazer login fornecendo suas credenciais (nome de usuário e senha) e usa a autenticação WebAuthn.<br><br>Então o sistema verifica a validade das credenciais fornecidas pelo usuário<br><br>E o sistema solicita a autenticação WebAuthn.<br><br>E o usuário realiza a autenticação com sucesso usando sua chave WebAuthn | O usuário é autenticado com sucesso e pode acessar o sistema. |
| 12. | Dado um usuário registrado no sistema com uma chave WebAuthn associada<br><br>Quando o usuário tenta fazer login fornecendo suas credenciais (nome de usuário e senha) e usa a autenticação WebAuthn<br><br>Então o sistema verifica a validade das credenciais fornecidas pelo usuário<br><br>E o sistema solicita a autenticação WebAuthn | A autenticação falha e o usuário não pode acessar o sistema. |

| | | |
|---|---|---|
| | E o usuário tenta realizar a autenticação usando uma chave WebAuthn inválida ou inexistente<br><br>Então o sistema recusa o acesso ao usuário | |
| 13. | Dado um usuário registrado no sistema com uma chave WebAuthn associada<br>Quando o usuário tenta fazer login fornecendo credenciais inválidas (nome de usuário ou senha incorretos)<br>Então o sistema verifica a validade das credenciais fornecidas pelo usuário<br>E o sistema detecta que as credenciais são inválidas | A autenticação falha devido às credenciais inválidas e o usuário não pode acessar o sistema. |

Fonte: elaborada pelo autor.

## 6.RESULTADOS

O microsserviço desenvolvido foi dividido em camadas seguindo o padrão definido pela arquitetura limpa. As camadas testadas foram: Domain, Data, Infra e Presentation.

A camada Domain é composta pela camada Model, onde foram desenvolvidos e testados componentes de domínio para a criação de um usuário, abstraindo a regra de negócio. A cobertura de instruções foi de 153 de 166, totalizando uma cobertura de 92%, e de ramificações foram cobertas 4 de 4, totalizando um percentual de cobertura para essa camada de 92%.A camada Data é responsável por centralizar a regra de negócio da aplicação, sendo composta pelas camadas Usecases e Handlers. A cobertura de instruções da camada de Usecases foi de 220 de 220, totalizando uma cobertura de 100%, e de ramificações foi de 30 de 36, dessa maneira com uma cobertura de 83%. A camada de Handlers possui a cobertura de instruções de 180 de 182, com um total de 98%, e de ramificações foi 2 de 2, com um total de 100%.A camada de Infra ficou responsável por lidar com o gerenciamento de instruções do banco de dados, ela é

composta pela camada Repository, a cobertura de testes de instruções foi de 390 de 447 com uma porcentagem de 87%, os testes de ramificações foi de 3 de 4, totalizando 75% de cobertura. Na camada de Presentation foram criados testes de integração. Esta camada expõe as requisições HTTP, é composta por outras duas camadas, Users e Authentication. A camada de Users é responsável por expor as requisições relacionadas ao fluxo de criação e atualização de usuários, a cobertura de instruções foi de 172 de 191 totalizando a porcentagem de 91% e de ramificações foi 20 de 20 totalizando 100%. A camada de Authentication é responsável por expor as requisições de autenticação e autorização, os testes de cobertura de instruções foi de 353 de 369 resultando na porcentagem de 95%, e de ramificação de 18 de 22 com a porcentagem de 81%.

A cobertura total da aplicação foi de aproximadamente 91% para instruções e 82% para ramificações, mantendo uma porcentagem aceitável acima de 80%. Foram realizados 76 testes, todos passaram sem falhas. Com uma cobertura de teste abrangente, as chances de erros nas instruções do serviço são baixas, tornando os cenários de falha menos prováveis.

## 7.CONCLUSÃO

O serviço concluído passou por testes abrangentes, tanto manuais quanto automatizados com o uso do Junit. O gerenciamento de identidades e acessos foi implementado e verificado em cada componente do sistema. A utilização do Mockito permitiu simular o comportamento da aplicação em cenários reais, aumentando a confiança na criação de um microserviço capaz de disponibilizar funcionalidades básicas de gerenciamento de identidades e acessos, alcançando assim o objetivo do trabalho.

A adoção da Arquitetura Limpa teve um impacto positivo nos resultados obtidos. A separação do sistema em camadas facilitou a implementação de testes e a adição de novas funcionalidades. Quando utilizadas funções recém desenvolvidas, poucos ajustes nos testes foram necessários e, quando ocorreram, foram simples de serem feitos. Esses ajustes normalmente foram ocasionados por cenários não validados que resultaram em exceções inesperadas.

A partir deste trabalho, espera-se continuar a implementação das dependências da aplicação, possibilitando seu uso como um serviço para gerenciamento de acessos e identidades. Além disso, sugere-se a adição de novas funcionalidades, aproveitando a arquitetura planejada para facilitar a manutenção e escalabilidade do sistema. Esse trabalho também serve como base para futuros estudos sobre arquitetura de sistemas similares.

REFERÊNCIAS


DANTAS, Francisco C; CASILLO, Dr Leonardo A; NETO, Francisco Milton M. **Uma Proposta de Arquitetura de Software Limpa baseada em Micros serviços**. 2021. [*s. l.*], 2021.

FONTES, Edison Luiz Goncalves. Segurança da Informação - O Usuário Faz a Diferença. São Paulo: Saraiva, 2006.

FOWLER, Susan. **Micro serviços prontos para produção.** São Paulo: Novatec Editora Ltda, 2017.

GLUU INC. **Gluu**. [*S. l.*], 2021. Disponível em: https://gluu.org/. Acesso em: 23 abr. 2022.

LEWIS, James; LEWIS, Martin. **Microservices**. [*S. l.*], 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 11 mar. 2022.

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT. [*S. l.*], 2001. Disponível em: https://agilemanifesto.org/. Acesso em: 3 jun. 2022.

MARTIN, Robert C. **Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software.** Rio de Janeiro: Alta Books, 2019.

NARDELLI, Cleber. Segurança da Informação e LGPD Aplicado no Desenvolvimento de Software. 2021. - Escola Regional de Engenharia de Software (ERES), [s. l.], 2021. Disponível em: https://sol.sbc.org.br/index.php/eres/article/view/18462/18295. Acesso em: 18 abr. 2022.

ORION SERVICES. [*S. l.*], [*s. d.*]. Disponível em: https://orion-services.dev/. Acesso em: 11 jun. 2022.

PEFFERS, Ken *et al.* Design Science Research Process: A Model for Producing and Presenting Information Systems Research. [*s. l.*], 2020. Disponível em: https://arxiv.org/abs/2006.02763. Acesso em: 3 jun. 2022.

RED HAT. **Keycloak**. [*S. l.*], 2021. Disponível em: https://www.keycloak.org/. Acesso em: 23 abr. 2022.

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

LEAL, Rhand . Controle de acesso da ISO 27001: Uso de autenticação pode dois fatores. 2017.

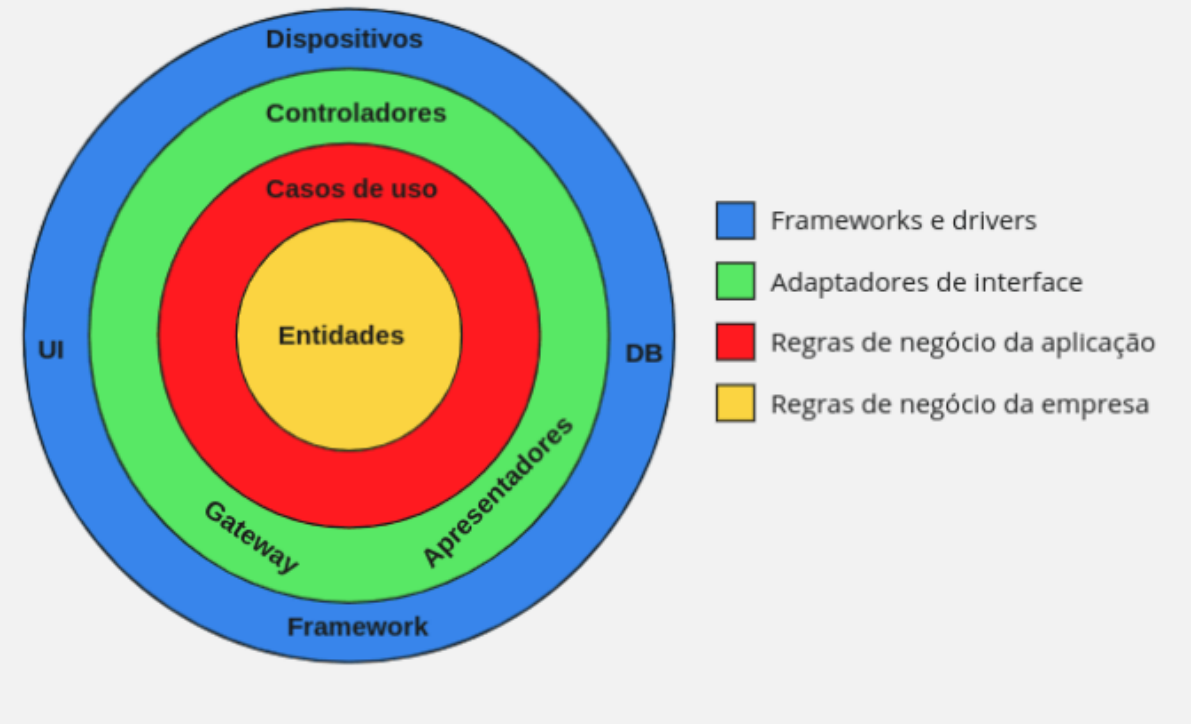
Disponível em: https://advisera.com/27001academy/pt-br/blog/2017/01/17/como-a-autenticacao-por-dois-fatores-apoia-a-conformidade-com-os-controles-de-acesso-da-iso-27001/. Acesso em: 15 jun. 2023

QUARKUS. [*S. l.*], [*s. d.*]. Disponível em: https://quarkus.io/. Acesso em: 05 dez. 2022.

JUNIT. [*S. l.*], [*s. d.*]. Disponível em: https://junit.org/junit5/. Acesso em: 06 dez. 2022.

## APÊNDICE 1 – FIGURA SOBRE ARQUITETURA LIMPA

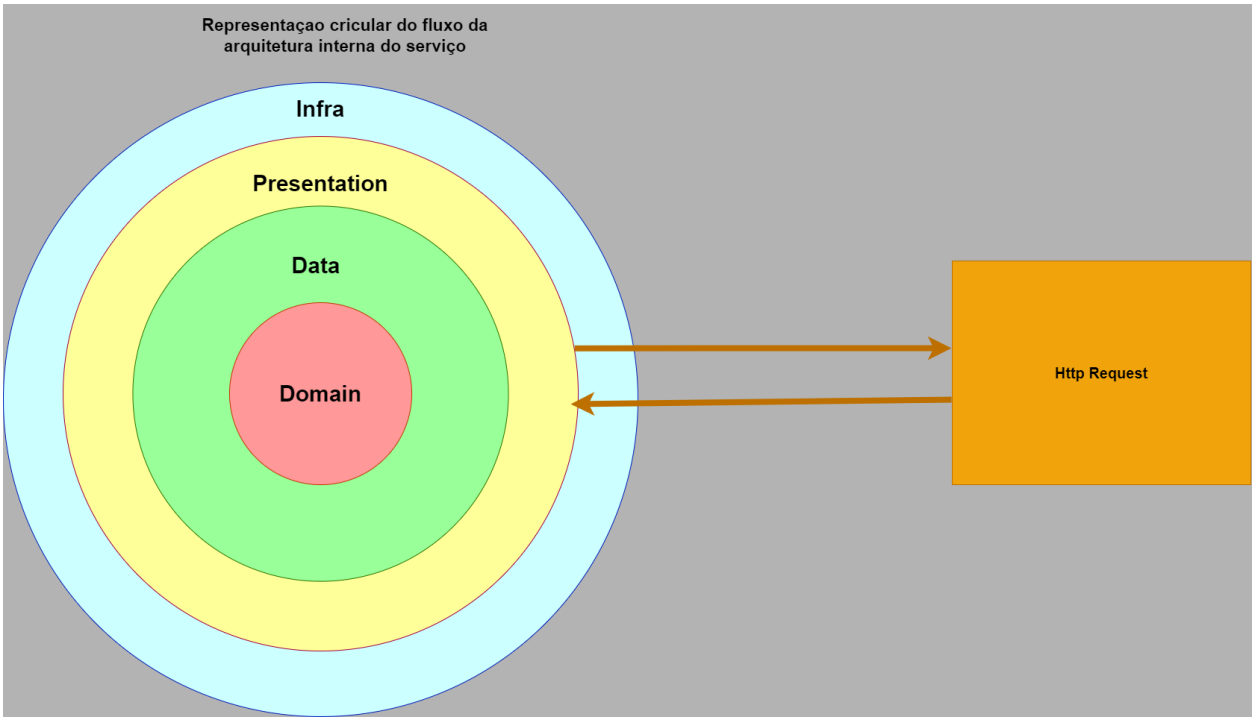
**Figura 1. Arquitetura limpa**




Fonte: Adaptada de Martin (2018, p.203)

## APÊNDICE 2A –ARQUITETURA INTERNA DO MICROSERVIÇO.

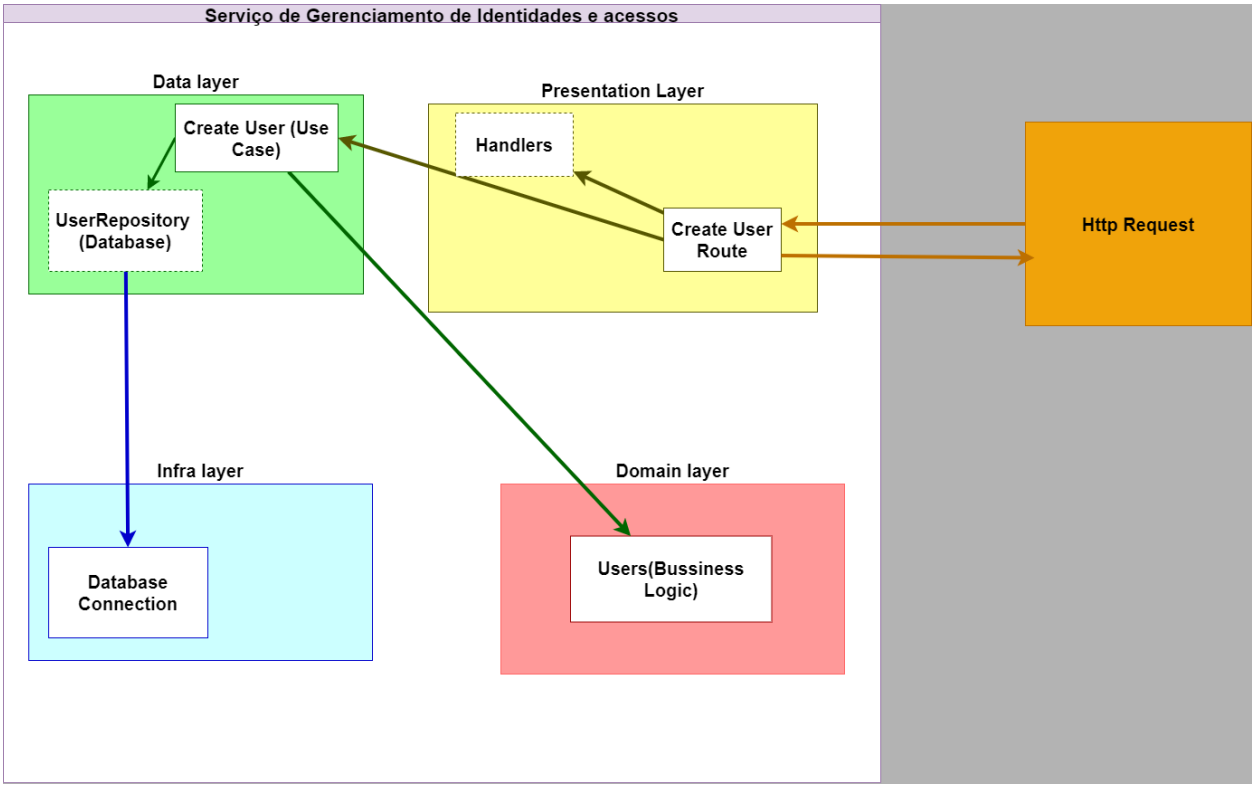
**Figura 2. Arquitetura Interna Circular do Sistema Proposto**




Fonte: elaborada pelo autor.

**APÊNDICE 2B –ARQUITETURA INTERNA DO MICROSERVIÇO.**

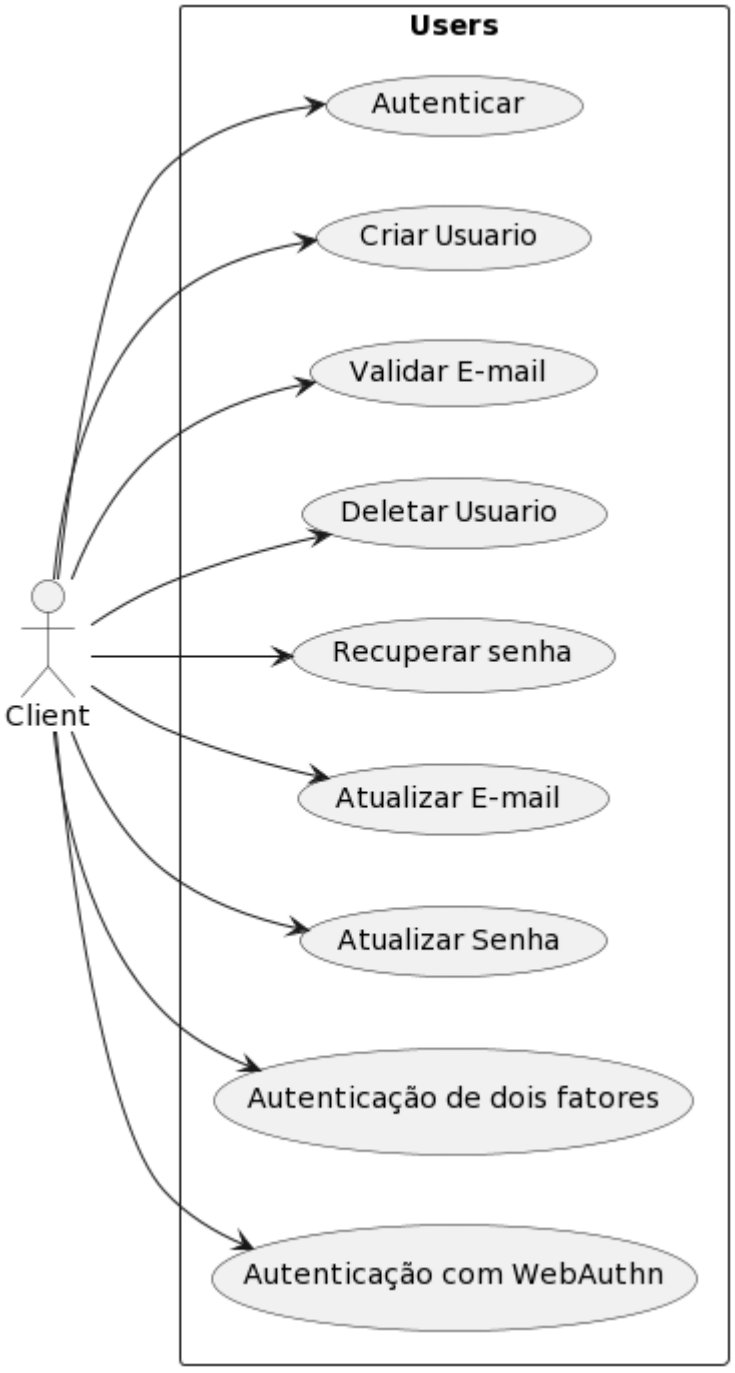
**Figura 3. Arquitetura Interna do Sistema Proposto**




Fonte: elaborada pelo autor.

**APÊNDICE 3 – CASOS DE USO**

**Figura 4. Diagrama de casos de uso**




Fonte: elaborada pelo autor.