

UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC CENTRO DE  
EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAÍ - CEA VI ENGENHARIA  
DE SOFTWARE

MAYARA MONTEIRO DE SOUZA

SISTEMA ORIENTADO À SERVIÇOS – SISTEMA DE CHAMADOS

IBIRAM

A

2025

1



## SUMÁRIO

OBJETIVO GERAL .....	3
VISÃO GERAL DO SISTEMA.....	4
1. USUÁRIOS DO SISTEMA.....	5
2. REQUISITOS FUNCIONAIS (RF) – NÍVEL USUÁRIO .....	7
3. MODELOS DO SISTEMA.....	8
REQUISITOS FUNCIONAIS (RF) – NÍVEL SISTEMA (DETALHAMENTO DE CASOS DE USO) .....	10
4. REQUISITOS NÃO FUNCIONAIS (RNF) .....	15
5. USER STORIES.....	17
6. FERRAMENTAS ALM .....	22
7. API GATEWAY .....	22
8. COMUNICAÇÃO.....	22
9. ARQUITETURA .....	22
10. DETALHAMENTO DOS MICROSSERVIÇOS .....	22
11. PADRÕES DE PROJETO GoF .....	22
12. TESTES .....	22
13. DOCUMENTAÇÃO.....	22

## OBJETIVO GERAL

Este documento tem como objetivo detalhar a arquitetura, funcionamento e estrutura de desenvolvimento do Sistema de Chamados Orientado a Serviços, cujo propósito é permitir que funcionários de uma organização abram, acompanhem e interajam com tickets de suporte interno, garantindo rastreabilidade, comunicação e organização entre setores.

O sistema foi construído com uma abordagem orientada a microserviços, garantindo modularidade, escalabilidade e autonomia entre os domínios:

- Autenticação (Auth Service)
- Diretório Organizacional (Directory Service)
- Gestão de Chamados (Tickets Service)
- API Gateway (Nginx)

O projeto está disponibilizado na plataforma GitHub. Para acessar o repositório de armazenamento da aplicação basta acessar o link: Arquitetura de Microserviços — Sistema de Chamados

## VISÃO GERAL DO SISTEMA

O Sistema de Chamados desenvolvido neste projeto tem como objetivo centralizar e organizar demandas internas de suporte dentro de uma organização. Ele permite que usuários registrem chamados, acompanhem suas solicitações e interajam com equipes responsáveis pelo atendimento. O sistema foi construído seguindo uma arquitetura orientada a microserviços, garantindo modularidade, escalabilidade, facilidade de manutenção e independência entre os domínios.

A solução é composta por três microsserviços principais — Auth Service, Directory Service e Tickets Service — que se comunicam por meio de um API Gateway, o qual atua como ponto único de entrada para todas as requisições. Esse modelo reduz o acoplamento entre serviços e simplifica o consumo da API por aplicações clientes.

Do ponto de vista funcional, o sistema contempla três tipos de usuários: Administrador, Usuário e Suporte.

Cada perfil possui permissões distintas:

Administrador: possui acesso total ao sistema, podendo gerenciar usuários, setores e chamados.

Usuário: pode criar chamados e acompanhar suas próprias solicitações.

Suporte: possui acesso completo a todos os chamados, podendo visualizar, comentar e atualizar o status das solicitações.

O fluxo geral do sistema inicia com a autenticação do usuário, que obtém um token JWT para acessar as rotas protegidas. Em seguida, o usuário pode criar chamados, visualizar informações ou, no caso do suporte, executar atendimentos diretamente no Tickets Service. Todas as interações seguem o padrão REST, utilizando métodos HTTP e dados no formato JSON.

A implementação baseada em microserviços permite que cada módulo seja atualizado ou escalado de forma independente, oferecendo maior flexibilidade e robustez à aplicação. Além disso, a presença de health-checks para cada serviço possibilita monitoramento contínuo da disponibilidade e estado da solução.

Em resumo, o sistema fornece uma plataforma estruturada para gestão de chamados internos, com controle de acesso por perfil, registro completo de histórico, fluxo de atendimento e comunicação entre setores — tudo isso sustentado por uma arquitetura moderna e escalável.

## 1. USUÁRIOS DO SISTEMA

Esta seção descreve os usuários do sistema, seus papéis e suas respectivas responsabilidades dentro do sistema de chamados.

### **Administrador**

O Administrador é o perfil com maior nível de permissão. Ele pode criar, atualizar, listar e visualizar qualquer informação do sistema, incluindo usuários, setores e chamados. Tem controle total sobre operações administrativas e também sobre os chamados, podendo comentar, alterar status e visualizar todo o histórico de uma solicitação.

### **Usuário**

O Usuário é o solicitante de atendimento. Ele pode criar novos chamados e visualizar apenas aqueles em que é o autor. Não possui permissão para alterar status ou comentar chamados. Seu papel é exclusivamente abrir solicitações e acompanhar seu andamento.

### **Suporte**

O Supor te é responsável por atender demandas internas. Ele tem acesso completo a tudo que envolve chamados: pode visualizar todos, alterar status, adicionar comentários, consultar histórico e acompanhar solicitações independentemente do setor ou do solicitante. Sua atuação está restrita à esfera de chamados, não podendo gerenciar setores ou usuários.

**Tabela – Usuários do Sistema**

Usuário	Descrição do papel	Funções
<b>Administrador</b>	Possui acesso completo a todas as informações e funcionalidades do sistema.	<ul style="list-style-type: none"> <li>- Cadastrar setores</li> <li>- Listar setores</li> <li>- Cadastrar usuários</li> <li>- Listar usuários</li> <li>- Criar chamados</li> <li>- Listar todos os chamados</li> <li>- Consultar detalhes de chamados</li> <li>- Adicionar comentários</li> <li>- Atualizar status</li> <li>- Visualizar todos os comentários</li> </ul>
<b>Usuário</b>	Pode criar novos chamados e acompanhar apenas as suas próprias solicitações.	<ul style="list-style-type: none"> <li>- Criar chamados</li> <li>- Listar seus próprios chamados</li> <li>- Consultar detalhes dos próprios chamados</li> <li>- Visualizar comentários dos seus chamados</li> </ul>
<b>Suporte</b>	Possui acesso a todas as informações relacionadas a chamados e comentários, sem restrição.	<ul style="list-style-type: none"> <li>- Listar todos os chamados</li> <li>- Consultar detalhes de qualquer chamado</li> <li>- Criar comentários em qualquer chamado</li> <li>- Visualizar comentários</li> <li>- Atualizar status de qualquer chamado</li> </ul>

## 2. REQUISITOS FUNCIONAIS (RF) – NÍVEL USUÁRIO

Esta seção apresenta os requisitos funcionais do sistema no nível de usuário, descrevendo o que cada ator pode realizar dentro da aplicação. Os RFs abaixo representam diretamente as operações que os usuários – Administrador, Usuário e Suporte – são capazes de executar conforme suas permissões no sistema.

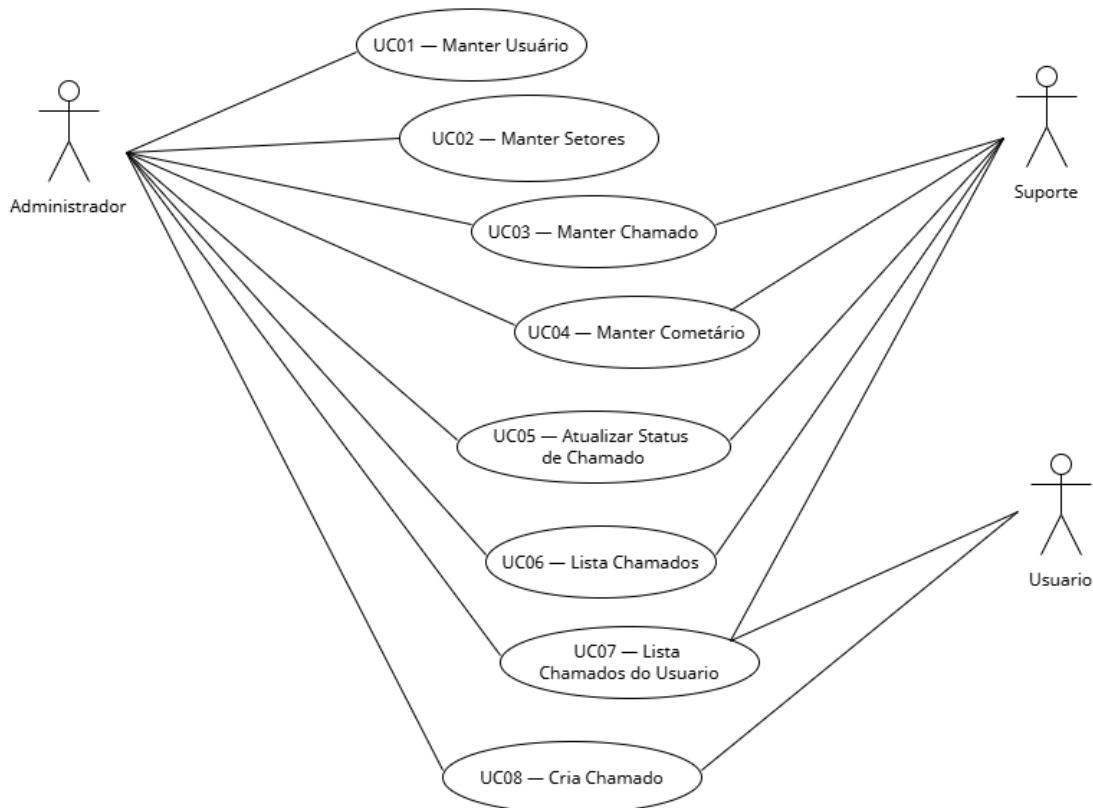
ID	Descrição
RF01	O sistema deve permitir registrar usuários.
RF02	O sistema deve permitir autenticar usuários e gerar tokens JWT.
RF03	O sistema deve ter dois tipos de usuário: Administrador, Suporte e Usuário.
RF04	O sistema deve permitir listar usuários
RF05	O sistema deve permitir listar setores.
RF06	O sistema deve permitir cadastrar setores.
RF07	O sistema deve permitir listar comentários de um chamado.
RF08	O sistema deve permitir buscar detalhes de um chamado.
RF09	O sistema deve permitir listar todos os chamados.
RF10	O sistema deve permitir adicionar comentários a um chamado.
RF11	O sistema deve permitir criar chamados.
RF12	O sistema deve permitir listar chamados do usuário autenticado.

### 3. MODELOS DO SISTEMA

Esta seção apresenta os principais modelos utilizados para representar graficamente o comportamento, estrutura e funcionalidades do Sistema de Chamados. Os diagramas auxiliam na compreensão da dinâmica do sistema, seus fluxos internos, seus estados e seus relacionamentos estruturais.

#### 3.1 DIAGRAMA DE CASOS DE USO

O diagrama de casos de uso representa, de forma visual, as funcionalidades disponibilizadas pelo sistema e como cada ator interage com essas funcionalidades. Os atores envolvidos são:



#### 3.2 DIAGRAMA DE ATIVIDADES

O diagrama de atividades modela o fluxo de execução das principais ações do sistema, descritas de forma sequencial. Ele mostra o passo a passo desde a criação de um chamado até sua tratativa pelo suporte.

#### 3.3 DIAGRAMA DE ESTADOS

O diagrama de estados representa a evolução do chamado ao longo do seu ciclo de vida. Cada mudança de status do chamado corresponde a uma transição entre estados, e cada transição é acionada por uma ação realizada pelo Suporte ou Administrador.

#### 3.4 DIAGRAMA DE CLASSES

O diagrama de classes representa a estrutura estática do sistema, com suas entidades principais, atributos, relacionamentos e responsabilidades.

## REQUISITOS FUNCIONAIS (RF) – NÍVEL SISTEMA (DETALHAMENTO DE CASOS DE USO)

Os requisitos funcionais representam as funções que o sistema deve obrigatoriamente oferecer aos seus usuários, descrevendo **comportamentos, operações permitidas e interações** entre usuários e microserviços.

O conjunto de RFs abaixo está diretamente associado aos **Casos de Uso (UC)** especificados na seção anterior, garantindo uma rastreabilidade clara entre requisitos, funcionalidades e atores envolvidos.

A seguir apresenta-se o detalhamento dos Requisitos Funcionais de acordo com cada Caso de Uso:

<b>Caso de Uso</b>	UC01 – Manter Usuário
<b>Descrição</b>	Este caso de uso descreve as etapas para o administrador registrar um novo usuário, autenticar usuários no sistema e consultar os dados do usuário autenticado.
<b>Requisitos</b>	RF01, RF02
<b>Atores</b>	Administrador, Usuário, Suporte
<b>Pré-Condições</b>	US02 – Usuário já cadastrado (para login e consulta);
<b>Pós-Condições</b>	US01 – Usuário registrado; US02 – Usuário autenticado; US03 – Dados do usuário retornados.
<b>Fluxo Base</b>	US01 – Registrar Usuário.
<b>Fluxos Alternativos</b>	US02 – Realizar login; US03 – Consultar dados do usuário autenticado.
<b>Fluxos de Exceção</b>	-

<b>Caso de Uso</b>	UC02 – Manter Setores
<b>Descrição</b>	Este caso de uso descreve as etapas para o administrador cadastrar e listar setores organizacionais que poderão ser utilizados como destino dos chamados.
<b>Requisitos</b>	RF04
<b>Atores</b>	Administrador

<b>Pré-Condições</b>	Administrador logado ao sistema;
<b>Pós-Condições</b>	US04 – Setor cadastrado; US05 – Setores listados.
<b>Fluxo Base</b>	US04 – Cadastrar setor.
<b>Fluxos Alternativos</b>	US05 – Listar setores.
<b>Fluxos de Exceção</b>	-

<b>Caso de Uso</b>	UC03 – Manter Chamado
<b>Descrição</b>	Este caso de uso descreve as etapas para o administrador criar chamados em nome de usuários, atualizar informações gerais do chamado (como setor de destino, título ou descrição) e consultar detalhes de um chamado.
<b>Requisitos</b>	RF05, RF06, RF10
<b>Atores</b>	Administrador, Suporte
<b>Pré-Condições</b>	US02 – Usuário (Administrador ou Suporte) autenticado; US11 – Chamado previamente criado.
<b>Pós-Condições</b>	US07 – Detalhes do chamado retornados.
<b>Fluxo Base</b>	US07 – Buscar detalhes de um chamado.
<b>Fluxos Alternativos</b>	-
<b>Fluxos de Exceção</b>	-

<b>Caso de Uso</b>	UC04 – Manter Comentário
<b>Descrição</b>	Este caso de uso descreve as etapas para o suporte registrar comentários em um chamado existente, de modo a registrar tratativas, dúvidas e retornos ao usuário solicitante.
<b>Requisitos</b>	RF07, RF10
<b>Atores</b>	Suporte
<b>Pré-Condições</b>	US02 – Suporte autenticado; US11 – Chamado previamente criado.

<b>Pós-Condições</b>	US10 – Comentário adicionado ao chamado; US08 – Comentários do chamado listados.
<b>Fluxo Base</b>	US10 – Adicionar comentários a um chamado.
<b>Fluxos Alternativos</b>	US08 – Listar comentários de um chamado.
<b>Fluxos de Exceção</b>	-

<b>Caso de Uso</b>	<b>UC05 – Atualiza Status de Chamado</b>
<b>Descrição</b>	Este caso de uso descreve as etapas para o suporte atualizar o status de um chamado, por exemplo, de Aberto para Em Andamento, Aguardando Resposta, Resolvido ou Fechado.
<b>Requisitos</b>	RF08 ( <i>consulta dos detalhes do chamado antes da atualização de status</i> )
<b>Atores</b>	Suporte, Administrador
<b>Pré-Condições</b>	US02 – Suporte autenticado; US11 – Chamado previamente criado.
<b>Pós-Condições</b>	US09 – Status do chamado atualizado.
<b>Fluxo Base</b>	US09 – Atualizar status do chamado.
<b>Fluxos Alternativos</b>	-
<b>Fluxos de Exceção</b>	-

<b>Caso de Uso</b>	<b>UC06 – Lista Chamados</b>
<b>Descrição</b>	Descreve as etapas para o administrador e o suporte listarem todos os chamados cadastrados no sistema, a fim de acompanhar o volume, priorizar atendimentos e identificar chamados em aberto.
<b>Requisitos</b>	RF09

<b>Atores</b>	Administrador, Suporte
<b>Pré-Condições</b>	US02 – Usuário autenticado (Administrador ou Suporte).
<b>Pós-Condições</b>	US09 – Lista de chamados exibida.
<b>Fluxo Base</b>	US09 – Listar todos os chamados.
<b>Fluxos Alternativos</b>	US09 – Listar todos os chamados.
<b>Fluxos de Exceção</b>	-
<b>Caso de Uso</b>	UC07 – Lista Chamados do Usuário
<b>Descrição</b>	Este caso de uso descreve as etapas para o usuário visualizar apenas os chamados em que ele é o solicitante, permitindo acompanhar o andamento de suas solicitações.
<b>Requisitos</b>	RF12
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	US02 – Usuário autenticado.
<b>Pós-Condições</b>	US12 – Lista de chamados do usuário autenticado exibida.
<b>Fluxo Base</b>	US12 – Listar chamados do usuário autenticado.
<b>Fluxos Alternativos</b>	US07 – Buscar detalhes de um chamado do usuário; US08 – Listar comentários do chamado.
<b>Fluxos de Exceção</b>	-

<b>Caso de Uso</b>	UC08 – Cria Chamado
<b>Descrição</b>	Este caso de uso descreve as etapas para o usuário ou administrador criar um novo chamado, informando título, descrição, setor de destino e solicitante.

<b>Requisitos</b>	RF11
<b>Atores</b>	Usuário, Administrador
<b>Pré-Condições</b>	US02 – Usuário ou Administrador autenticado; US06 – Setor de destino cadastrado.
<b>Pós-Condições</b>	US11 – Chamado criado com status “Aberto”.
<b>Fluxo Base</b>	US13 – Criar chamado.
<b>Fluxos Alternativos</b>	US06 – Criar chamado pelo administrador em nome de outro usuário.
<b>Fluxos de Exceção</b>	-

## 4. REQUISITOS NÃO FUNCIONAIS (RNF)

Os requisitos não funcionais representam características essenciais relacionadas ao desempenho, segurança, arquitetura, confiabilidade e experiência de uso do sistema de chamados desenvolvido com microserviços. Diferente dos requisitos funcionais, que descrevem *o que* o sistema deve fazer, os requisitos não funcionais definem *como* o sistema deve operar, garantindo qualidade, consistência e boas práticas de desenvolvimento.

A seguir são apresentados os requisitos não funcionais organizados por categoria:

**Tabela de Requisitos Não Funcionais**

ID	Categoria	Descrição
RNF01	Arquitetura	O sistema deve ser desenvolvido utilizando arquitetura de microserviços.
RNF02	Comunicação	Os serviços devem se comunicar utilizando APIs REST com mensagens no formato JSON.
RNF03	Segurança	A autenticação deve ser realizada via tokens JWT em todas as rotas protegidas.
RNF04	Segurança	As senhas dos usuários devem ser armazenadas de forma criptografada em banco de dados.
RNF05	Persistência	O serviço de autenticação deve utilizar banco de dados relacional (por exemplo, PostgreSQL).
RNF06	Persistência	O serviço de chamados deve utilizar banco de dados adequado a registros de tickets (por exemplo, MongoDB).
RNF07	Desempenho	Cada requisição deve ter tempo de resposta de, no máximo, 3 segundos em condições normais de uso.
RNF08	Confiabilidade	Os serviços devem expor endpoints de health-check para verificação de disponibilidade.
RNF09	Escalabilidade	A solução deve permitir escalabilidade horizontal dos microserviços de forma independente.
RNF10	Documentação	As APIs dos microserviços devem ser documentadas utilizando Swagger ou ferramenta equivalente.
RNF11	Manutenibilidade	O código deve seguir princípios de separação de responsabilidades, facilitando manutenção e evolução.

RNF12	Usabilidade	As respostas da API devem retornar mensagens claras de erro e sucesso para facilitar o uso pelo cliente.
-------	-------------	--

## 5. USER STORIES

As histórias de usuário representam as operações disponibilizadas para os usuários do sistema. Elas são criadas a partir dos requisitos funcionais identificados e detalhados nos casos de uso, refletindo as necessidades reais dos atores envolvidos no processo.

Para este projeto, adotou-se um formato simplificado, composto por três perguntas principais:

- **Como:** quem executa a ação?
- **Eu quero:** o que a ação realiza?
- **Para:** qual o objetivo pretendido?

Após essa descrição inicial, são apresentados os **passos necessários** para execução da tarefa e seus **critérios de aceitação**, indicando o que é indispensável para considerar a história concluída.

A seguir, são listadas as histórias de usuário associadas a cada caso de uso do sistema.

### 6.1. US01 – Registrar Usuário

User Case: UC01

**Como:** Administrador

**Eu quero:** Registrar novos usuários no sistema

**Para:** Que eles possam acessar as funcionalidades conforme seu perfil de acesso

**Passos:**

1. Administrador acessa o sistema.
2. Navega até a opção “Cadastro de Usuário”.
3. O sistema exibe o formulário de cadastro.
4. O administrador informa os dados obrigatórios (nome, e-mail, senha, tipo e setor).
5. Clica no botão “Registrar Usuário”.
6. O sistema valida e salva os dados.
7. O sistema apresenta mensagem de sucesso.

**Critérios de aceitação:**

1. Todos os campos obrigatórios devem estar preenchidos.
2. O e-mail informado não pode estar previamente cadastrado.
3. A senha deve ser armazenada de forma criptografada.
4. O tipo de usuário selecionado deve ser válido (Admin, Suporte ou Usuário).

## 6.2. US02 – Realizar Login

User Case: UC01

**Como:** Usuário, Administrador ou Suporte

**Eu quero:** Fazer login no sistema

**Para:** Acessar as funcionalidades conforme meu perfil

**Passos:**

1. Usuário acessa a tela de login.
2. Informa e-mail e senha cadastrados.
3. Clica em “Entrar”.
4. O sistema valida as credenciais.
5. Se válidas, o sistema gera e retorna um token JWT.
6. O usuário é direcionado ao menu principal.

**Critérios de aceitação:**

1. As credenciais informadas devem ser válidas.
2. O token JWT deve ser retornado para acesso às rotas protegidas.
3. Caso os dados estejam incorretos, o sistema deve exibir mensagem de erro.

## 6.3. US03 – Consultar Dados do Usuário Autenticado

User Case: UC01

**Como:** Usuário autenticado

**Eu quero:** Consultar minhas informações

**Para:** Verificar meus dados e permissões no sistema

**Passos:**

1. Usuário envia requisição para obter seus dados.
2. O token JWT é enviado no cabeçalho Authorization.
3. O sistema valida o token.
4. O sistema retorna os dados do usuário autenticado.

**Critérios de aceitação:**

1. A requisição deve conter um token válido.
2. O sistema só deve retornar dados do próprio usuário.

## 6.4. US04 – Listar Setores

User Case: UC02

**Como:** Administrador

**Eu quero:** Ver a lista de setores cadastrados

**Para:** Consultar e organizar o direcionamento dos chamados

**Passos:**

1. Administrador seleciona “Listar Setores”.
2. O sistema consulta o Directory Service.
3. O sistema exibe todos os setores cadastrados.

**Critérios de aceitação:**

1. A lista deve exibir todos os setores ativos.
2. O sistema deve retornar erro caso o serviço esteja indisponível.

## 6.5. US05 – Cadastrar Setor

User Case: UC02

**Como:** Administrador

**Eu quero:** Cadastrar novos setores

**Para:** Estruturar o fluxo organizacional dos chamados

**Passos:**

1. Administrador acessa “Cadastrar Setor”.
2. Informa o nome do setor.
3. Clica em “Confirmar”.
4. Sistema valida o nome e registra o setor.
5. Sistema exibe mensagem de sucesso.

**Critérios de aceitação:**

1. O nome do setor deve ser único.
2. O campo nome é obrigatório.

## 6.6. US06 – Criar Chamado

User Case: UC08

**Como:** Usuário ou Administrador

**Eu quero:** Criar um novo chamado

**Para:** Registrar uma solicitação ao setor responsável

**Passos:**

1. Usuário acessa a área “Criar Chamado”.
2. Preenche o título, descrição e setor de destino.
3. O sistema identifica o solicitante pelo token.
4. O sistema salva o chamado com status “Aberto”.
5. Exibe mensagem de sucesso.

**Critérios de aceitação:**

1. O setor informado deve existir.
2. Todos os campos obrigatórios devem ser preenchidos.

## 6.7. US07 – Consultar Detalhes do Chamado

User Case: UC03 / UC06 / UC07

**Como:** Usuário, Suporte ou Administrador

**Eu quero:** Ver detalhes completos do chamado

**Para:** Acompanhar seu conteúdo e histórico

**Passos:**

1. Usuário seleciona um chamado.
2. Sistema consulta o Tickets Service.
3. Sistema retorna título, descrição, status, datas e histórico.

**Critérios de aceitação:**

1. O ID informado deve existir.
2. Usuário comum só pode ver chamados que ele criou.

## 6.8. US08 – Listar Comentários de um Chamado

User Case: UC04

**Como:** Suporte, Administrador ou Usuário solicitante

**Eu quero:** Ver todos os comentários do chamado

**Para:** Acompanhar o histórico das tratativas

**Passos:**

1. Selecionar um chamado.
2. Sistema busca os comentários associados.
3. Sistema exibe o histórico completo.

**Critérios de aceitação:**

1. O chamado deve existir.
2. Caso não existam comentários, deve retornar lista vazia.

## 6.8. US08 – Listar Comentários de um Chamado

User Case: UC04

**Como:** Suporte, Administrador ou Usuário solicitante

**Eu quero:** Ver todos os comentários do chamado

**Para:** Acompanhar o histórico das tratativas

**Passos:**

1. Selecionar um chamado.
2. Sistema busca os comentários associados.
3. Sistema exibe o histórico completo.

**Critérios de aceitação:**

1. O chamado deve existir.
2. Caso não existam comentários, deve retornar lista vazia.

## 6.10. US10 – Atualizar Status do Chamado

User Case: UC05

**Como:** Suporte

**Eu quero:** Atualizar o status de um chamado

**Para:** Informar o andamento do atendimento

**Passos:**

1. Suporte acessa o chamado.
2. Seleciona o novo status.
3. Clica em “Atualizar”.
4. Sistema salva a alteração.
5. Registra a data da última atualização.

**Critérios de aceitação:**

1. O status selecionado deve ser válido.
2. O chamado deve existir.

## 6.11. US11 – Listar Todos os Chamados

User Case: UC06

**Como:** Suporte ou Administrador

**Eu quero:** Visualizar todos os chamados do sistema

**Para:** Organizar prioridades e acompanhar o volume de demandas

**Passos:**

1. Usuário acessa “Listar Chamados”.
2. O sistema consulta o Tickets Service.
3. Exibe todos os chamados.

**Critérios de aceitação:**

1. Deve listar todos os chamados ativos.
2. O sistema deve retornar erro caso o serviço esteja indisponível.

## 6.12. US12 – Listar Chamados do Usuário

User Case: UC07

**Como:** Usuário

**Eu quero:** Ver apenas os meus chamados

**Para:** Acompanhar minhas próprias solicitações

**Passos:**

1. Usuário acessa "Meus Chamados".
2. Sistema identifica o usuário pelo token.
3. Exibe apenas chamados criados por aquele usuário.

**Critérios de aceitação:**

1. O usuário deve estar autenticado.
2. Não deve listar chamados de outros usuários.

## 6. API GATEWAY

O API Gateway funciona como ponto único de entrada para o *Sistema de Chamados com Microserviços*. Em vez de cada cliente se comunicar diretamente com cada serviço (Auth, Directory e Tickets), todas as requisições passam pelo gateway, que se encarrega de encaminhá-las para o microserviço correto.

No projeto atual, o API Gateway está configurado para atender na URL base:

- <http://localhost:8081>

A partir dessa URL, o gateway roteia as requisições para os serviços internos de acordo com o caminho (path) da requisição.

### 4.1 Papel do API Gateway no sistema

O API Gateway tem as seguintes responsabilidades principais:

- **Roteamento de requisições:** encaminha cada requisição recebida para o microserviço adequado com base no path (/auth, /directory, /tickets).
- **Ponto único de acesso:** centraliza o consumo da API, simplificando o uso pelo cliente (Postman, front-end, etc.).
- **Isolamento dos serviços:** oculta portas e detalhes internos dos microserviços, expondo apenas o endereço do gateway.
- **Monitoramento básico:** disponibiliza endpoints de *health-check* de cada serviço para verificação de disponibilidade.

### 4.2 Mapeamento de rotas

A Tabela 1 apresenta o mapeamento das rotas expostas pelo API Gateway com base na collection do Postman.

**Tabela 1 – Rotas do API Gateway**

Caminho (Gateway)	Método	Descrição	Serviço de destino
<a href="http://localhost:8081/auth/health">http://localhost:8081/auth/health</a>	GET	Verifica a saúde do Auth Service	Auth Service
<a href="http://localhost:8081/auth/register">http://localhost:8081/auth/register</a>	POST	Registra um novo usuário	Auth Service
<a href="http://localhost:8081/auth/login">http://localhost:8081/auth/login</a>	POST	Autentica usuário e gera token JWT	Auth Service

<a href="http://localhost:8081/auth/me">http://localhost:8081/auth/me</a>	GET	Retorna informações do usuário autenticado	Auth Service
<a href="http://localhost:8081/auth/user-types">http://localhost:8081/auth/user-types</a>	GET	Lista os tipos de usuário permitidos	Auth Service
<a href="http://localhost:8081/directory/health">http://localhost:8081/directory/health</a>	GET	Verifica a saúde do Directory Service	Directory Service
<a href="http://localhost:8081/directory/sectors">http://localhost:8081/directory/sectors</a>	GET	Lista setores cadastrados	Directory Service
<a href="http://localhost:8081/directory/sectors">http://localhost:8081/directory/sectors</a>	POST	Cadastra novo setor	Directory Service
<a href="http://localhost:8081/directory/employees">http://localhost:8081/directory/employees</a>	GET	Lista usuários/funcionários	Directory Service
<a href="http://localhost:8081/tickets/health">http://localhost:8081/tickets/health</a>	GET	Verifica a saúde do Tickets Service	Tickets Service
<a href="http://localhost:8081/tickets">http://localhost:8081/tickets</a>	POST	Cria novo chamado	Tickets Service
<a href="http://localhost:8081/tickets">http://localhost:8081/tickets</a>	GET	Lista todos os chamados	Tickets Service
<a href="http://localhost:8081/tickets/me">http://localhost:8081/tickets/me</a>	GET	Lista chamados do usuário autenticado	Tickets Service
<a href="http://localhost:8081/tickets/{id}">http://localhost:8081/tickets/{id}</a>	GET	Busca detalhes de um chamado por ID	Tickets Service
<a href="http://localhost:8081/tickets/{id}/status">http://localhost:8081/tickets/{id}/status</a>	PATCH	Atualiza o status de um chamado	Tickets Service
<a href="http://localhost:8081/tickets/{id}/comments">http://localhost:8081/tickets/{id}/comments</a>	POST	Adiciona comentário a um chamado	Tickets Service
<a href="http://localhost:8081/tickets/{id}/comments">http://localhost:8081/tickets/{id}/comments</a>	GET	Lista comentários de um chamado	Tickets Service

### 4.3 Integração com os microserviços

Cada microserviço (Auth, Directory e Tickets) é responsável por uma parte específica do domínio do sistema. O API Gateway atua como uma **fachada**, recebendo as requisições externas em `localhost:8081` e encaminhando internamente para:

- **Auth Service** – autenticação e gestão de usuários;
- **Directory Service** – gestão de setores e listagem de funcionários;
- **Tickets Service** – criação, consulta e atualização de chamados e comentários.

Dessa forma, o cliente (front-end ou Postman) não precisa conhecer as portas internas ou endereços individuais dos serviços, lidando apenas com o gateway.

#### 4.4 Benefícios do uso de API Gateway

O uso de um API Gateway no sistema de chamados traz alguns benefícios importantes:

- **Simplificação do consumo da API:** o cliente utiliza uma única URL base, independentemente da quantidade de serviços internos.
- **Facilidade de manutenção:** mudanças em portas ou detalhes internos de cada serviço não impactam o cliente, desde que o contrato do gateway seja mantido.
- **Evolução gradual:** novos serviços podem ser adicionados ao ecossistema e expostos via gateway sem alterar a interface pública principal.
- **Centralização de políticas:** no futuro, podem ser adicionadas políticas de autenticação, rate limiting e logs diretamente no gateway.

## 7. COMUNICAÇÃO

A comunicação aplicada ao sistema de chamados é do tipo **REST (Representational State Transfer)**, um estilo arquitetural amplamente utilizado na criação de APIs para sistemas distribuídos. Nesse modelo, a troca de informações entre o cliente, o **API Gateway** e os microsserviços (**Auth, Directory e Tickets**) é realizada por meio de requisições HTTP utilizando URL's bem definidas, métodos padrão (GET, POST, PATCH) e códigos de status para indicar o resultado das operações.

No contexto deste trabalho, o **API Gateway** expõe uma URL base (<http://localhost:8081>) e, a partir dela, direciona as requisições para cada serviço de acordo com o caminho:

- /auth para autenticação e gestão de usuários;
- /directory para setores e funcionários;
- /tickets para criação, consulta e atualização de chamados.

As mensagens trafegadas entre o cliente e os microsserviços utilizam o formato **JSON**, o que simplifica a serialização e desserialização de dados e torna a integração mais padronizada. Além disso, o modelo REST adotado é **stateless**, ou seja, o servidor não mantém estado de sessão entre as requisições, exigindo que o cliente envie as informações necessárias a cada chamada, incluindo o **token JWT** no cabeçalho **Authorization** nas rotas protegidas.

Entre as vantagens da utilização da comunicação REST neste sistema, destacam-se:

- **Simplicidade de aplicação**, uma vez que as operações são mapeadas diretamente em métodos HTTP;
- **Escalabilidade**, permitindo que cada microsserviço seja escalado horizontalmente de forma independente, conforme a demanda;
- **Independência de tecnologia**, já que a API pode ser consumida por qualquer cliente capaz de realizar requisições HTTP e interpretar JSON;
- **Padronização de respostas**, por meio do uso consistente de códigos de status HTTP (como 200, 201, 400, 401, 404), facilitando o tratamento de erros e sucessos no consumo da API.

## 8. ARQUITETURA

A arquitetura adotada no Sistema de Chamados é baseada em **microsserviços**, com comunicação via HTTP e uso de um **API Gateway** como ponto central de entrada. Cada microsserviço possui sua própria responsabilidade, banco de dados e lógica de negócio isolada, permitindo escalabilidade, separação de domínios e manutenção independente.

Além da separação entre serviços, a aplicação também utiliza o conceito de **arquitetura em camadas**, implementada de forma lógica em todos os microsserviços e fisicamente organizada no Tickets Service.

A seguir, descreve-se a arquitetura completa com base no código-fonte atual.

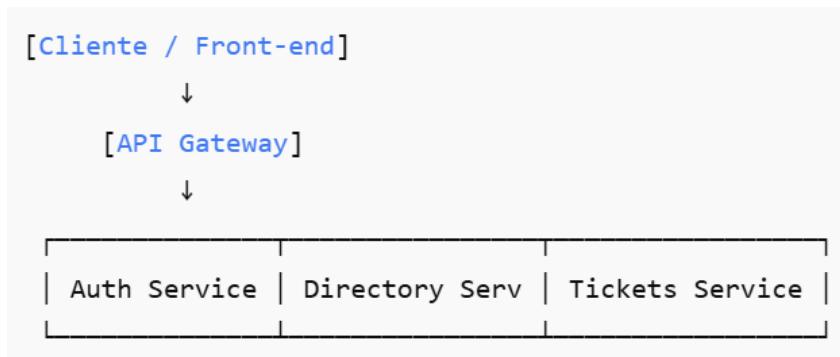
### 4.1 Visão Geral da Arquitetura

A solução é composta pelos seguintes serviços:

- **API Gateway**: responsável por rotear todas as requisições para os microsserviços corretos.
- **Auth Service**: responsável pelo cadastro, autenticação, geração de tokens JWT e validação de usuários.
- **Directory Service**: gerencia setores e funcionários.

- **Tickets Service:** responsável pela criação, atualização, listagem e fluxo de comentários de chamados.

Fluxo geral:



Cada serviço possui sua própria base de dados contida dentro da pasta `/sql`, o que reforça a independência entre os componentes.

## 4.2 Arquitetura em Camadas Aplicada

Embora a organização física das camadas varie entre os microsserviços, todos seguem os princípios básicos de separação de responsabilidades, dividindo-se em:

- **Camada de Apresentação:** tratamento das requisições HTTP.
- **Camada de Aplicação:** regras do caso de uso.
- **Camada de Domínio:** regras de negócio do contexto específico.
- **Camada de Infraestrutura:** acesso ao banco de dados e dependências técnicas.

A seguir, cada microsserviço é detalhado conforme está implementado no projeto.

## 4.3 API Gateway

```

api-gateway
├── package.json
└── server.js
  
```

 Código: `api-gateway/server.js` 🚀 O API Gateway não possui múltiplas camadas internas estruturadas em pastas devido à sua simplicidade. Ainda assim, ele segue responsabilidades distintas.

### Camada de Apresentação

Recebe todas as requisições externas vindas do cliente.

### Camada de Aplicação

- Gerencia o roteamento para os serviços:
  - /auth → Auth Service
  - /directory → Directory Service
  - /tickets → Tickets Service
- Aplica middlewares como logs e CORS.

### Camada de Infraestrutura

Utiliza http-proxy-middleware para encaminhamento das requisições.

## 4.4 Auth Service

```

auth-service
  └── sql
    ├── db.js
    ├── Dockerfile
    ├── migracao_auth.sql
    ├── package-lock.json
    ├── package.json
    └── server.js
  
```

 **Código principal:** auth-service/server.js

 **Banco de dados:** auth-service/db.js + arquivos SQL

Neste serviço, as camadas existem **de forma lógica**, porém implementadas no mesmo arquivo.

### Camada de Apresentação

- Define endpoints:
  - POST /register
  - POST /login
  - GET /me
  - GET /user-types

### Camada de Aplicação

- Regras de autenticação:
  - Validação de login
  - Criptografia de senha
  - Criação de usuário
  - Geração e validação de token JWT
- Sincronização automática com o Directory Service durante o cadastro.

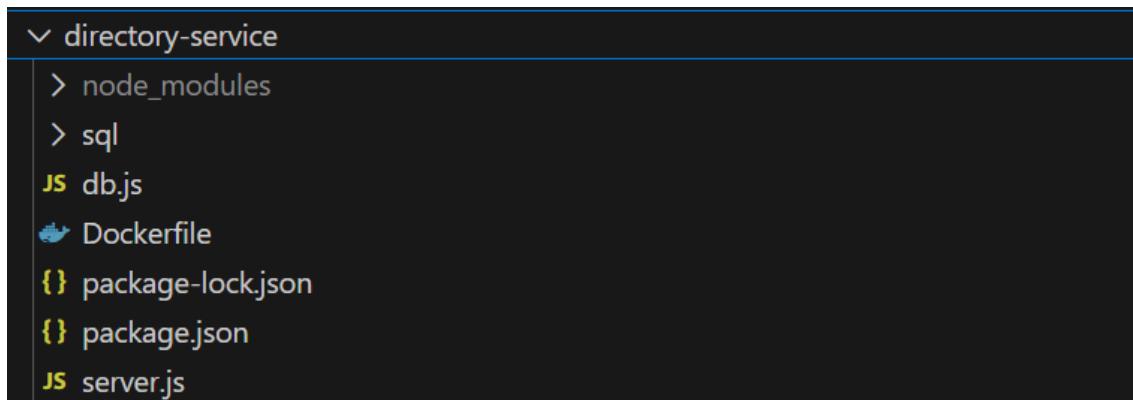
### Camada de Domínio

- Lógica de usuário:
  - Campos obrigatórios
  - Regras de permissão
  - Tipos de usuário (ADMIN, SUPORTE, USUARIO)

### Camada de Infraestrutura

- Conexão com banco SQL (db.js)
- Execução de queries
- Hash de senha com bcrypt
- JWT para autenticação

## 4.5 Directory Service



```
directory-service
  node_modules
  sql
  db.js
  Dockerfile
  package-lock.json
  package.json
  server.js
```

📁 Código principal: directory-service/server.js

📁 Banco: directory-service/db.js + scripts SQL

Assim como o Auth, este serviço utiliza camadas lógicas consolidadas em um único arquivo.

### Camada de Apresentação

- Endpoints:
  - POST /sectors
  - GET /sectors
  - POST /employees
  - GET /employees

### Camada de Aplicação

- Cadastro e listagem de setores
- Cadastro e listagem de funcionários
- Validações básicas como nome obrigatório

### Camada de Domínio

- Regras associadas a:
  - Setores
  - Funcionários

### Camada de Infraestrutura

- Conexão com banco via db.js
- Execução dos scripts SQL
- Queries diretas para manipulação de dados

### 4.6 Tickets Service

```
▽ tickets-service
  > controllers
  > repositories
  > services
  > sql
  JS db.js
  Dockerfile
  {} package-lock.json
  {} package.json
  JS server.js
```

Este é o microsserviço mais completo e **único que implementa fisicamente a arquitetura em camadas**.

### Camada de Apresentação

#### 📁 controllers/ticketsController.js

- Recebe e trata requisições HTTP
- Endpoints:
  - Criar ticket
  - Listar tickets
  - Atualizar status
  - Buscar por ID
  - Comentários do ticket

### Camada de Aplicação

#### 📁 services/ticketsService.js

- Regras de negócio dos casos de uso:
  - Criar tickets
  - Registrar comentários

- Atualizar status
- Registrar data de atualização
- Verificar dados enviados
- Listar tickets do usuário autenticado

### Camada de Domínio

- Conceitos representados:
  - Ticket
  - Comment
- Implementados diretamente no service, sem classes isoladas

Regras implementadas:

- Status válidos: *Aberto, Em Andamento, Aguardando Resposta, Resolvido, Fechado*
- Atualização automática de `dataUltimaAtualizacao`

### Camada de Infraestrutura

📁 repositories/ticketsRepo.js  
📁 db.js  
📁 sql/001\_init.sql

- Execução de consultas SQL
- Persistência de tickets e comentários
- Criação de tabelas no banco de dados
- Conexão de baixo nível com a base

## 4.7 Considerações Finais da Arquitetura

Com base no código implementado, observa-se que:

- O **API Gateway** atua somente como intermediador e não possui lógica de domínio.
- O **Auth Service** e o **Directory Service** utilizam camadas lógicas, porém integradas em arquivos únicos.
- O **Tickets Service** implementa arquitetura em camadas de forma completa, com divisão clara entre controller, serviço, repositório e banco.
- A separação entre serviços e bancos diferentes reforça a independência dos componentes.

## 9. DETALHAMENTO DOS MICROSSERVIÇOS

Esta seção apresenta o detalhamento dos microsserviços que compõem o Sistema de Chamados, descrevendo a linguagem utilizada, estrutura de armazenamento, arquitetura aplicada e os endpoints disponibilizados por cada serviço.

A solução foi desenvolvida utilizando o estilo arquitetural de **microsserviços**, com comunicação via **API REST**, e os serviços executam de forma independente, cada um com seu próprio banco de dados e responsabilidades bem definidas.

### 10.1. Microserviço de Autenticação (Auth Service)

O microserviço de autenticação é responsável pelo gerenciamento de usuários, autenticação, geração de token JWT e validação de permissões dentro do sistema. Este serviço centraliza todo o fluxo de segurança, garantindo que apenas usuários autenticados e autorizados accessem os recursos dos demais microsserviços.

#### Linguagem e Tecnologias Utilizadas

- **Linguagem:** Node.js
- **Framework:** Express
- **Criptografia de senhas:** Biblioteca BCrypt
- **Autenticação:** JWT (JSON Web Token) para geração e validação de tokens
- **Banco de Dados:** PostgreSQL (via queries SQL diretas)
- **Comunicação:** API REST
- **Função adicional:** Ao registrar um usuário, o Auth Service envia automaticamente dados para o Directory Service, garantindo sincronização entre as informações de usuário e funcionário.

#### Estrutura de Armazenamento

O banco contém a tabela:

- **users**
  - id
  - name
  - email
  - password\_hash
  - tipo\_usuario (ADMIN, SUPORTE, USUARIO)
  - setor
  - Cargo

## Responsabilidades Principais

- Cadastro de novos usuários
- Autenticação via email e senha
- Geração de token JWT
- Validação do usuário autenticado
- Sincronização de funcionário com o Directory Service

## Endpoints Disponíveis

- **POST** [\*\*/auth/register\*\*](#)  
Cadastra um novo usuário com senha criptografada e registra também no Directory Service.
- **POST** [\*\*/auth/login\*\*](#)  
Realiza autenticação e retorna um token JWT válido.
- **GET** [\*\*/auth/me\*\*](#)  
Retorna as informações do usuário com base no token enviado no cabeçalho.
- **GET** [\*\*/auth/user-types\*\*](#)  
Lista os tipos de usuário disponíveis no sistema.

## 10.2. Microserviço de Diretório (Directory Service)

O Directory Service é responsável pela gestão de setores e funcionários cadastrados no sistema. Ele atua como uma camada de organização interna, permitindo que chamados sejam direcionados ao setor correto.

### Linguagem e Tecnologias Utilizadas

- **Linguagem:** Node.js
- **Framework:** Express
- **Banco:** PostgreSQL
- **Comunicação:** API REST
- **Arquitetura:** Camadas lógicas consolidadas em um único arquivo (controller + regras de domínio)

### Estrutura de Armazenamento

O banco contém as tabelas:

- **sectors**
  - id
  - name
- **employees**
  - id
  - name
  - email
  - setor
  - Cargo

## Responsabilidades do Serviço

- Cadastrar setores
- Listar setores
- Listar e cadastrar funcionários (automático quando um usuário é criado no Auth Service)

## Endpoints Disponíveis

- **POST** [\*\*/directory/sectors\*\*](#)  
Cria um novo setor no sistema.
- **GET** [\*\*/directory/sectors\*\*](#)  
Retorna todos os setores cadastrados.
- **GET** [\*\*/directory/employees\*\*](#)  
Lista todos os funcionários existentes.
- **POST** [\*\*/directory/employees\*\*](#)  
Armazena um novo funcionário (geralmente chamado automaticamente pelo Auth Service).

### 10.3. Microserviço de Chamados (Tickets Service)

O Tickets Service centraliza todo o fluxo de chamados, desde a abertura até a resolução, incluindo comentários, histórico de interações e atualização de status.

Este é o microsserviço que aplica **arquitetura em camadas de forma física**, com pastas separadas para controller, service e repositório.

#### Linguagem e Tecnologias Utilizadas

- **Linguagem:** Node.js
- **Framework:** Express
- **Banco:** PostgreSQL
- **Arquitetura:** MVC em camadas (Controller → Service → Repository)
- **Comunicação:** API REST
- **Validação de acesso:** Token JWT enviado via API Gateway

#### Estrutura de Pastas (implementada fisicamente)

```
tickets-service/
  └── controllers
  └── services
  └── repositories
  └── db.js
  └── sql/
    └── server.js
```

#### Estrutura de Armazenamento

O banco contém:

- **tickets**
  - id
  - titulo
  - descricao
  - status
  - setor\_destino
  - solicitante\_id
  - data\_abertura
  - data\_ultima\_atualizacao

- **comments**
  - id
  - ticket\_id
  - autor\_id
  - mensagem
  - data\_criacao

## Status permitidos

- Aberto
- Em Andamento
- Aguardando Resposta
- Resolvido
- Fechado

## Responsabilidades Principais

- Abrir chamados
- Listar chamados (todos, por ID ou apenas do usuário autenticado)
- Atualizar status
- Registrar comentários
- Organizar histórico de interações
- Registrar datas de atualização automaticamente

## Endpoints Disponíveis

### **Chamados**

- **POST** /tickets  
Criação de um novo ticket.
- **GET** /tickets  
Listagem geral de tickets.
- **GET** /tickets/me  
Listagem dos tickets do usuário autenticado.
- **GET** /tickets/:id  
Busca de ticket por ID.

- **PATCH** `/tickets/:id/status`  
Atualização do status do ticket.

### **Comentários**

- **POST** `/tickets/:id/comments`  
Inserção de comentário em um ticket.
- **GET** `/tickets/:id/comments`  
Retorna todos os comentários vinculados ao ticket.

## 10.4. API Gateway

Embora não seja um microsserviço de regra de negócio, o API Gateway compõe a arquitetura ao centralizar as requisições e distribuir para os microsserviços.

### Linguagem e Tecnologias

- **Linguagem:** Node.js
- **Framework:** Express
- **Biblioteca principal:** http-proxy-middleware
- **Comunicação:** API REST

### Responsabilidades

- Roteamento para:
  - `/auth`
  - `/directory`
  - `/tickets`
- Aplicação de CORS
- Entrada única do sistema

### Endpoints

O Gateway não possui endpoints próprios.

Ele apenas encaminha:

- `/auth/*` → Auth Service
- `/directory/*` → Directory Service
- `/tickets/*` → Tickets Service

## 10. SOLID

Apesar de o projeto estar desenvolvido em Node.js, sem uso de orientação a objetos clássica, é possível identificar a aplicação prática dos princípios SOLID principalmente através da separação de responsabilidades entre microsserviços e dentro do Tickets Service.

### 12.1 Single Responsibility Principle (SRP)

Cada microsserviço possui uma única responsabilidade dentro do sistema:

- **Auth Service:** autenticação e gestão de usuários.
- **Directory Service:** setores e funcionários.
- **Tickets Service:** criação, atualização e consulta de chamados e comentários.

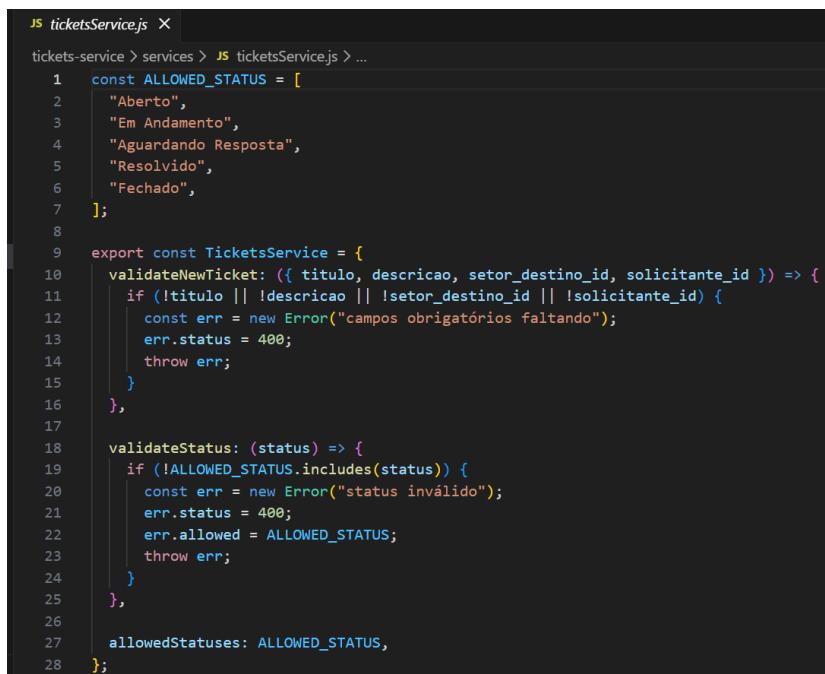
No **Tickets Service**, o SRP está ainda mais claro pela divisão física em camadas:

- `controllers/` → trata requisições HTTP
- `services/` → regras de negócio
- `repositories/` → acesso ao banco

Cada parte muda por um único motivo, alinhado ao SRP.

Arquivo: `tickets-service/services/ticketsService.js`

é possível observar o princípio de responsabilidade única (SRP), em que o `TicketsService` é responsável apenas por regras de validação de tickets e status, sem acessar diretamente banco de dados ou infraestrutura.



```

JS ticketsService.js ×
tickets-service > services > JS ticketsService.js > ...
1  const ALLOWED_STATUS = [
2    "Aberto",
3    "Em Andamento",
4    "Aguardando Resposta",
5    "Resolvido",
6    "Fechado",
7  ];
8
9  export const TicketsService = {
10    validateNewTicket: ({ titulo, descricao, setor_destino_id, solicitante_id }) => {
11      if (!titulo || !descricao || !setor_destino_id || !solicitante_id) {
12        const err = new Error("campos obrigatórios faltando");
13        err.status = 400;
14        throw err;
15      }
16    },
17
18    validateStatus: (status) => {
19      if (!ALLOWED_STATUS.includes(status)) {
20        const err = new Error("status inválido");
21        err.status = 400;
22        err.allowed = ALLOWED_STATUS;
23        throw err;
24      }
25    },
26
27    allowedStatuses: ALLOWED_STATUS,
28  };

```

### 12.2 Open/Closed Principle (OCP)

Os microsserviços podem ser estendidos sem precisar modificar sua estrutura principal.

Exemplos:

- O **controller** do Tickets Service continua igual mesmo que novas regras sejam adicionadas no `ticketsService.js`.
- Novas funcionalidades podem ser adicionadas criando novos métodos no `ticketsRepo.js`, sem alterar como os demais módulos funcionam.
- O **Traefik** permite adicionar novos microsserviços apenas criando novas regras de rota, sem alterar o código dos serviços existentes

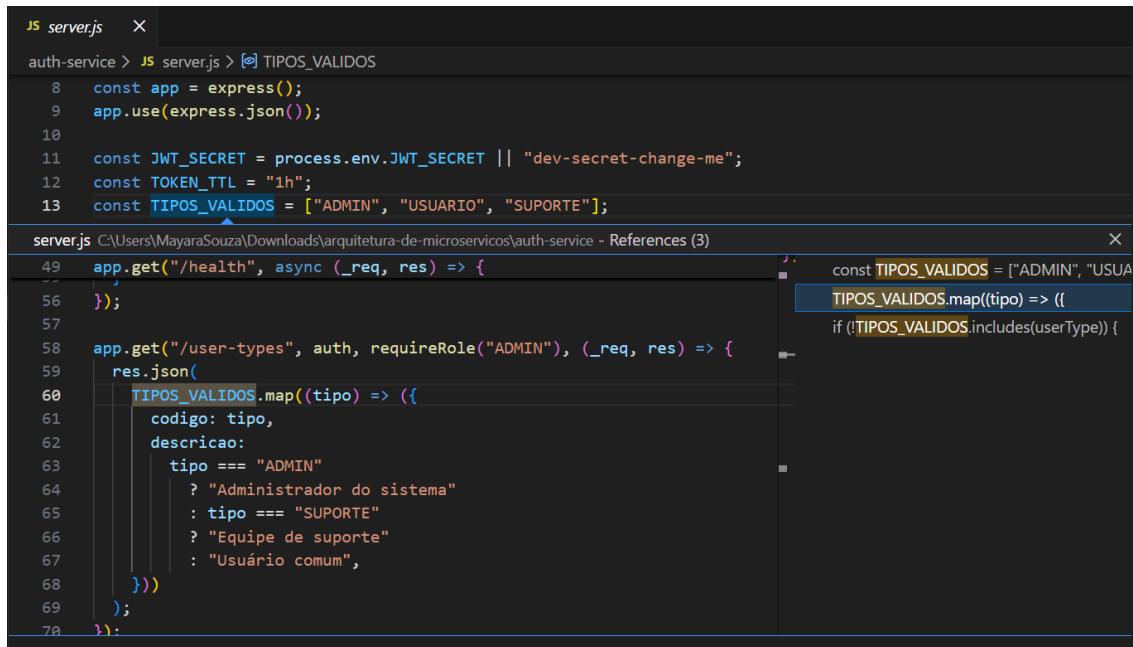
```
JS ticketsRepo.js X
tickets-service > repositories > JS ticketsRepo.js > ...
1 import { query } from "../db.js";
2
3 export const TicketsRepo = {
4   list: async () => {
5     const { rows } = await query(
6       `SELECT id, titulo, descricao, status, setor_destino_id, solicitante_id, created_at, updated_at
7       FROM chamado ORDER BY id`
8     );
9     return rows;
10   },
11
12   create: async ({ titulo, descricao, setor_destino_id, solicitante_id }) => {
13     const { rows } = await query(
14       `INSERT INTO chamado (titulo, descricao, setor_destino_id, solicitante_id)
15       VALUES ($1,$2,$3,$4)
16       RETURNING id, titulo, descricao, status, setor_destino_id, solicitante_id, created_at, updated_at`,
17       [titulo, descricao, Number(setor_destino_id), Number(solicitante_id)]
18     );
19     return rows[0];
20   },
21 }
```

## 12.3 Liskov Substitution Principle (LSP)

Embora o projeto não use herança, o LSP aparece de forma conceitual:

- Todos os microsserviços expõem **contratos estáveis** (endpoints).
- A implementação interna pode ser substituída (ex.: mudar o repositório, trocar SQL por ORM) sem quebrar os consumidores da API, desde que o contrato seja mantido.

De forma conceitual, o princípio de substituição de Liskov (LSP), pois o endpoint /user-types define um contratoável de resposta. A implementação interna pode ser alterada sem impactar os consumidores, desde que o contrato (estrutura de saída) seja mantido.



```

JS server.js  X
auth-service > JS server.js > [?] TIPOS_VALIDOS
  8   const app = express();
  9   app.use(express.json());
 10
 11  const JWT_SECRET = process.env.JWT_SECRET || "dev-secret-change-me";
 12  const TOKEN_TTL = "1h";
 13  const TIPOS_VALIDOS = ["ADMIN", "USUARIO", "SUPORTE"];
server.js C:\Users\MayaraSouza\Downloads\arquitetura-de-microservicos\auth-service - References (3)  X
 49  app.get("/health", async (_req, res) => {
 50    res.json({ status: "ok" });
 51  });
 52
 53  app.get("/user-types", auth, requireRole("ADMIN"), (_req, res) => {
 54    res.json(
 55      TIPOS_VALIDOS.map((tipo) => ({
 56        tipo,
 57        descricao: tipo === "ADMIN"
 58          ? "Administrador do sistema"
 59          : tipo === "SUPORTE"
 60          ? "Equipe de suporte"
 61          : "Usuário comum",
 62      })
 63    );
 64  });
 65}
 66
 67
 68
 69
 70

```

## 12.4 Interface Segregation Principle (ISP)

Cada microsserviço expõe apenas endpoints relacionados ao seu domínio específico:

- O cliente que usa Tickets não é obrigado a conhecer Auth ou Directory.
- Directory Service trata somente setores e funcionários.
- Auth Service trata apenas autenticação.

Isso evita “interfaces grandes” e segue diretamente o ISP.

### **auth-service:**

```

build: ./auth-service
labels:
- "traefik.enable=true"
- "traefik.http.routers.auth.rule=PathPrefix('/auth')"

```

### **directory-service:**

```

build: ./directory-service
labels:
- "traefik.enable=true"
- "traefik.http.routers.directory.rule=PathPrefix('/directory')"

```

### **tickets-service:**

```

build: ./tickets-service
labels:
- "traefik.enable=true"
- "traefik.http.routers.tickets.rule=PathPrefix('/tickets')"

```

## 11. TESTES

Foram implementados testes unitários para o TicketsService, utilizando o framework Jest, com foco na validação de criação de chamados e nos status permitidos. Os testes garantem que somente tickets com todos os campos obrigatórios são aceitos e que apenas os status definidos na regra de negócio são considerados válidos.

Os testes foram organizados utilizando `describe()` para agrupar funcionalidades e `it()` para representar cada cenário de teste. As verificações foram feitas com `expect()`, validando se os métodos do TicketsService se comportam corretamente.

Foram testados três pontos principais:

- Validação de campos obrigatórios (`validateNewTicket`);
- Validação dos status permitidos (`validateStatus`);
- Estrutura da lista de status aceitos (`allowedStatuses`).

Essa abordagem garante que as regras de negócio sejam aplicadas corretamente e que qualquer alteração futura que quebre essas regras seja detectada automaticamente.

## 12. DOCUMENTAÇÃO

No Sistema de Chamados, essa documentação é realizada utilizando o **Swagger**, uma das ferramentas mais consolidadas para documentação de APIs REST.

Microsserviço	URL via Gateway	URL direta
Tickets Service	<a href="http://localhost:8081/tickets/api-docs">http://localhost:8081/tickets/api-docs</a>	<a href="http://localhost:3003/api-docs">http://localhost:3003/api-docs</a>
Auth Service	–	–
Directory Service	–	–
API Gateway (Traefik)	–	–

## 13. Integração Contínua, Entrega Contínua e Configuração de Ambientes (Homolog e Produção)

Esta seção apresenta a implementação completa do pipeline de Integração Contínua e Entrega Contínua (CI/CD) do sistema de chamados desenvolvido com arquitetura de microserviços. O objetivo é garantir automação, padronização, controle de versões, rastreabilidade e segurança no processo de build, teste e disponibilização das imagens dos serviços. Além disso, são definidos os ambientes de execução (Desenvolvimento e Homologação) e a forma como cada um utiliza o pipeline.

## 1. Versionamento e Organização do Código

O repositório utiliza GitHub com o padrão:

- main → código estável
- feature/ → desenvolvimento de funcionalidades
- PR obrigatório para merge
- Validação automática do CI antes do merge

Essa estrutura garante rastreabilidade, histórico limpo e segurança na evolução do sistema.

## 2. Análise de Qualidade com SonarCloud

O projeto integra SonarCloud ao pipeline para garantir inspeção contínua de:

- Qualidade do código
- Code smells
- Vulnerabilidades
- Cobertura de testes
- Padrões de boas práticas

Arquivo utilizado: sonar-project.properties

## 3. Pipeline de Integração Contínua (CI)

O CI é executado automaticamente a cada:

- push na branch main
- pull request para main

O workflow realiza:

Etapas do CI

- Checkout do repositório
- Instalação das dependências
- Execução dos testes com Jest
- Geração da cobertura
- Upload da cobertura para SonarCloud
- Validação do “Quality Gate”

- Build da imagem Docker do serviço
- Push da imagem para o GitHub Container Registry

Arquivo utilizado: .github/workflows/ci.yml

#### 4. Construção e Empacotamento com Docker

Cada microserviço possui seu Dockerfile.

Exemplo da image do tickets-service:

```
FROM node:20 WORKDIR /app COPY package*.json . RUN npm install COPY .. EXPOSE 3000
CMD ["npm", "start"]
```

O CI gera automaticamente:

- ghcr.io/maymontsouza/tickets-service:latest
- ghcr.io/maymontsouza/tickets-service:

#### 5. Configuração dos Ambientes: Dev e Homolog

##### 5.1 Ambiente de Homolog

Utilizado para validação final antes de ir para Produção.

Neste ambiente são realizados:

- testes manuais avançados
- verificação de integrações
- validação de monitoramento e métricas
- análise de performance

O deploy em homolog usa:

Containers [Give feedback](#)

Container CPU usage 19.68% / 1200% (12 CPUs available)

Container memory usage 2.19GB / 7.41GB

[Show charts](#)

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	sonarqube	91e23f27d435	sonarqube:its	9000:9000	2.02%	54 minutes ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	arquitetura-de-microservicos	-	-	-	15.04%	11 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	auth-db-1	9b11f8518506	library/postgres:16	-	0.06%	11 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	directory-db-1	620fd58d799e	library/postgres:16	-	4.48%	12 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	tickets-db-1	bd925731df0c	library/postgres:16	-	8.96%	13 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	api-gateway-1	5ff79d468392	traefik:v3.0	8081:80	0%	14 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	directory-service-1	dd5caced87ed	arquitetura-de-microservicos-direc	-	0%	16 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	tickets-service-1	0e634e7e05dd	arquitetura-de-microservicos-ticke	-	0%	17 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	auth-service-1	416cae742bb5	arquitetura-de-microservicos-auth	-	0%	19 seconds ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	prometheus	438f3acc78be	prom/prometheus	9090:9090	0.44%	1 hour ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	grafana	594e0ea4948c	grafana/grafana:latest	3000:3000	1.11%	1 hour ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>

Showing 11 items

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	arquitetura-de-microservicos	-	-	-	0.88%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	auth-db-1	9b11f8518506	library/postgres:16	-	0.03%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	directory-db-1	620fd58d799e	library/postgres:16	-	0.03%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	tickets-db-1	bd925731df0c	library/postgres:16	-	0.03%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	api-gateway-1	5ff79d468392	traefik:v3.0	8081:80	0%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	directory-service-1	dd5caced87ed	arquitetura-de-microservicos-direc	-	0%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	tickets-service-1	0e634e7e05dd	arquitetura-de-microservicos-ticke	-	0%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	auth-service-1	416cae742bb5	arquitetura-de-microservicos-auth	-	0%	1 minute ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	prometheus	438f3acc78be	prom/prometheus	9090:9090	0%	1 hour ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>
<input type="checkbox"/>	grafana	594e0ea4948c	grafana/grafana:latest	3000:3000	0.79%	1 hour ago	<span style="color: green;">⋮</span> <span style="color: green;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span> <span style="color: red;">⋮</span>

Prometheus

Query Alerts Status > Target health

Select scrape pool Filter by target health Filter by endpoint or labels

prometheus

Endpoint	Labels	Last scrape	State
http://prometheus:9090/metrics	instance="prometheus:9090" job="prometheus"	2.58s ago	<span style="color: green;">UP</span>

1 / 1 up

