# DOCUMENTAÇÃO TÉCNICA DO PROJETO – API CHAMADOS TÉCNICOS

Título do Projeto: Sistema de Gerenciamento de Chamados Técnicos

Grupo: **:** Ellison Erik Robertty Pedroso de Oliveira, João Carlos Fava Filho e Jaques de Oliveira Vasconcelos \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 1. Objetivo do Projeto

O objetivo deste projeto é desenvolver uma API REST para o gerenciamento de chamados técnicos em uma organização. A aplicação foi construída com tecnologias modernas, como .NET 7, e permite o registro, acompanhamento e resolução de chamados relacionados a problemas de infraestrutura, software e sistemas corporativos. A API possibilita uma gestão eficiente entre usuários e técnicos, facilitando o controle e o histórico de atendimentos através de um fluxo de status bem definido.

## 2. Estrutura da Solução

2.1 Modelagem de Dados

A modelagem da aplicação foi definida com base nas entidades centrais do domínio, cada uma com responsabilidades específicas:

Entidades Principais:

• Chamado: Entidade central com atributos como título, descrição, status, urgência, modo de atendimento (remoto/presencial), tempo gasto, nota de atendimento e datas de abertura/fechamento

• Usuário: Representa quem abre os chamados, com nome, email e senha

• Técnico: Profissional responsável pelo atendimento, com nome e especialidade

• Problema: Categorização dos tipos de problemas, incluindo nome, descrição e solução comum

• Departamento: Organização dos problemas por área (TI, RH, Financeiro)

Fluxo de Status dos Chamados:

1. Aberto → Chamado criado pelo usuário

2. Em Andamento → Técnico atribuído e trabalhando

3. Resolvido → Técnico finalizou com nota e tempo gasto

4. Fechado → Chamado finalizado definitivamente

2.2 Integração da API com a Solução

A API foi estruturada utilizando Minimal APIs do ASP.NET Core, organizadas em grupos de rotas por entidade. A integração com banco de dados é realizada via Entity Framework Core com InMemory Database, facilitando desenvolvimento e testes. O sistema inclui validações através de Data Annotations e DTOs específicos para diferentes operações, garantindo integridade e segurança dos dados.

Características Técnicas:

• Uso de DTOs para transferência segura de dados

• Validações automáticas com Data Annotations

• Relacionamentos bem definidos entre entidades

• Inicialização automática com dados de exemplo

## 3. Endpoints da API

Chamados

Método Rota Descrição

GET /api/chamados Retorna todos os chamados com detalhes completos

GET /api/chamados/{id} Retorna um chamado específico por ID

POST /api/chamados Cria um novo chamado

DELETE /api/chamados/{id}/delete Exclui um chamado por ID

PUT /api/chamados/{id}/atender Atribui técnico e muda status para "Em Andamento"

PUT /api/chamados/{id}/resolver Registra resolução e muda status para "Resolvido"

PUT /api/chamados/{id}/fechar Finaliza chamado com status "Fechado"

Usuários

Método Rota Descrição

GET /api/usuarios Lista todos os usuários

GET /api/usuarios/{id} Retorna usuário específico por ID

POST /api/usuarios Cria novo usuário (com validação de email único)

GET /api/usuarios/{id}/chamados Lista chamados abertos pelo usuário

Técnicos

Método Rota Descrição

GET /api/tecnicos Lista todos os técnicos

GET /api/tecnicos/{id} Retorna técnico específico por ID

GET /api/tecnicos/disponiveis Lista técnicos com menos de 5 chamados abertos

GET /api/tecnicos/{id}/chamados Lista todos os chamados do técnico

GET /api/tecnicos/{id}/chamados/abertos Lista apenas chamados abertos do técnico

## 4. Organização do Código

O projeto está estruturado seguindo princípios de separação de responsabilidades:

Estrutura de Diretórios:

• Models/ – Entidades do domínio (Chamado.cs, Usuario.cs, Tecnico.cs, Problema.cs, Departamento.cs)

• DTOs/ – Objetos de transferência de dados com validações

o ChamadoDTO.cs – Para visualização completa

o ChamadoCreateDTO.cs – Para criação com validações

o ChamadoUpdateDTO.cs – Para resolução de chamados

o AtenderChamadoDTO.cs – Para atribuição de técnicos

• Data/ – Camada de acesso a dados

o AppDbContext.cs – Contexto Entity Framework

o DbInitializer.cs – Inicialização automática com dados de exemplo

o DataStore.cs – Alternativa em memória para testes

• Routes/ – Organização das rotas por funcionalidade

o ChamadosRoutes.cs – Endpoints de chamados

o UsuariosRoutes.cs – Endpoints de usuários

o TecnicosRoutes.cs – Endpoints de técnicos

• Program.cs – Configuração da aplicação e pipeline

## 5. Justificativa Técnica

Arquitetura e Design

A arquitetura foi projetada priorizando modularidade, manutenibilidade e escalabilidade. A escolha por Minimal APIs oferece performance superior e código mais conciso, enquanto a separação em arquivos de rotas facilita manutenção e evolução.

Modelagem Orientada ao Domínio

O uso de entidades fortemente tipadas com relacionamentos bem definidos garante integridade referencial. Os DTOs proporcionam uma camada de abstração que protege o modelo interno e permite validações específicas para cada operação.

Qualidade e Robustez:

• Validações automáticas através de Data Annotations

• Tratamento de erros com retornos HTTP apropriados

• Dados de exemplo ricos para facilitar testes (10 usuários, 10 técnicos, 10 problemas categorizados)

• Fluxo controlado de estados dos chamados

Escalabilidade

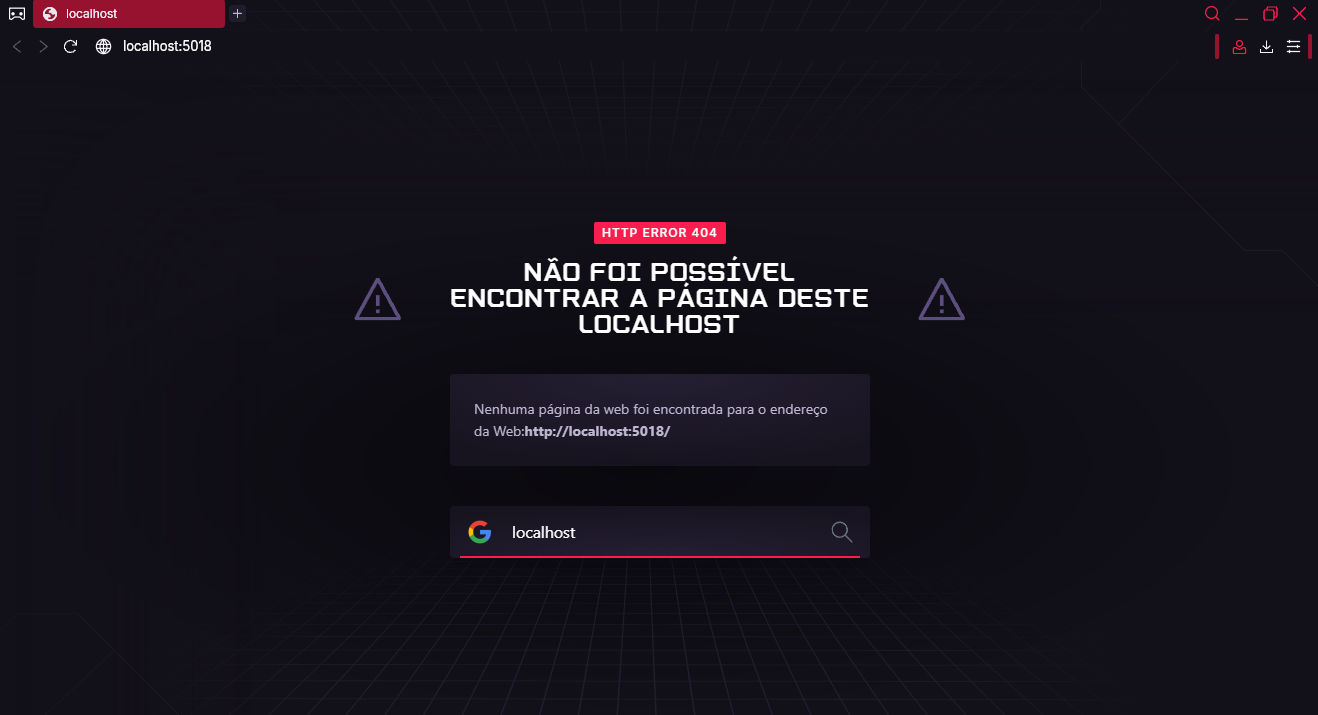
A base InMemory facilita desenvolvimento e testes, mas pode ser facilmente substituída por SQL Server, PostgreSQL ou outros SGBDs. A estrutura modular permite futuras expansões como autenticação, autorização, notificações e dashboards administrativos.

Endpoint Utilitário

O sistema inclui um endpoint /ver-dados para visualização completa da base de dados, facilitando desenvolvimento e debugging.

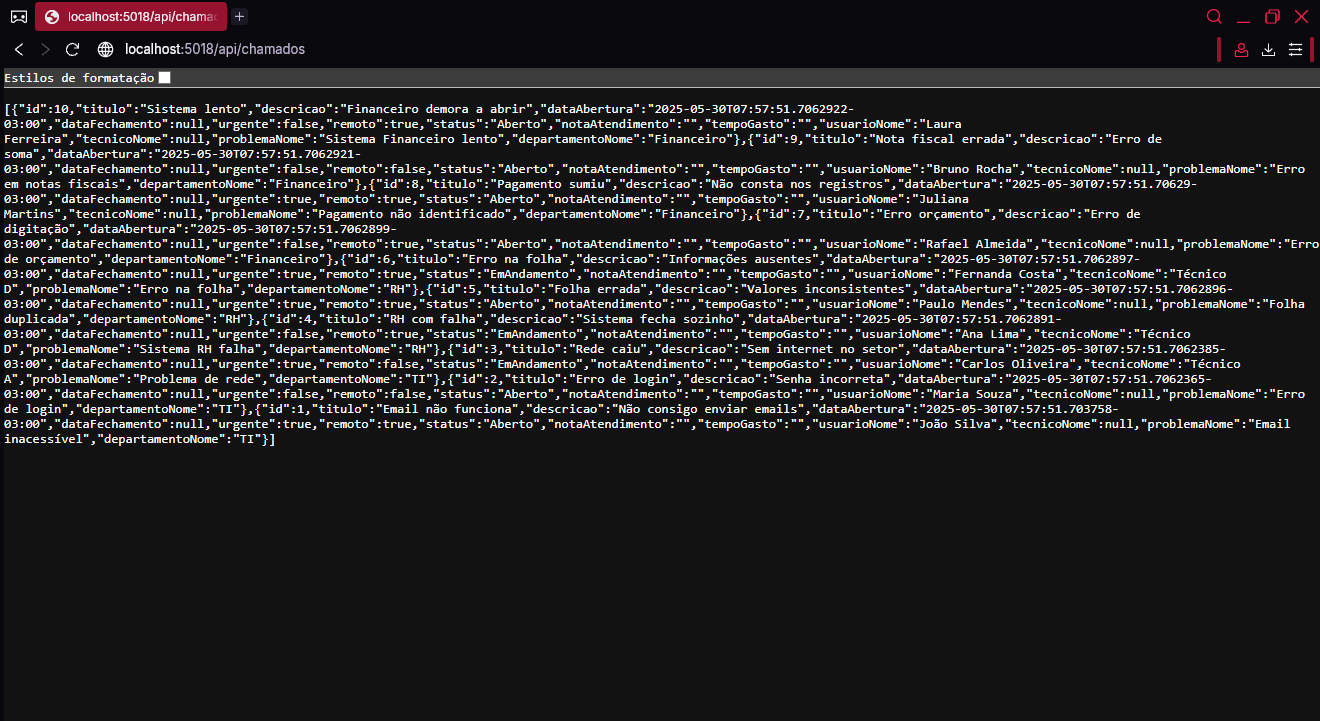
## 6. Seção de Testes

Figura 1 – LocalHost sem rotas



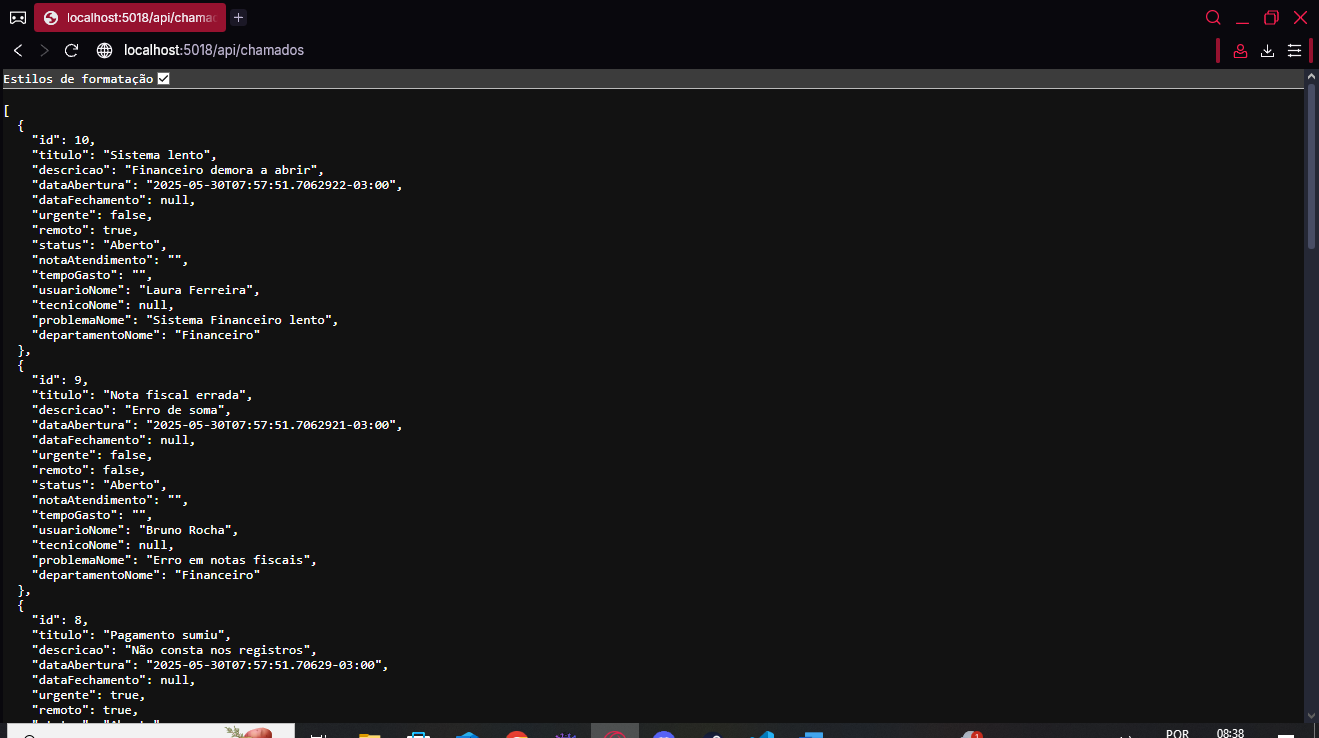
Fonte: Autoral

Figura 2 – LocalHost com a rota Get



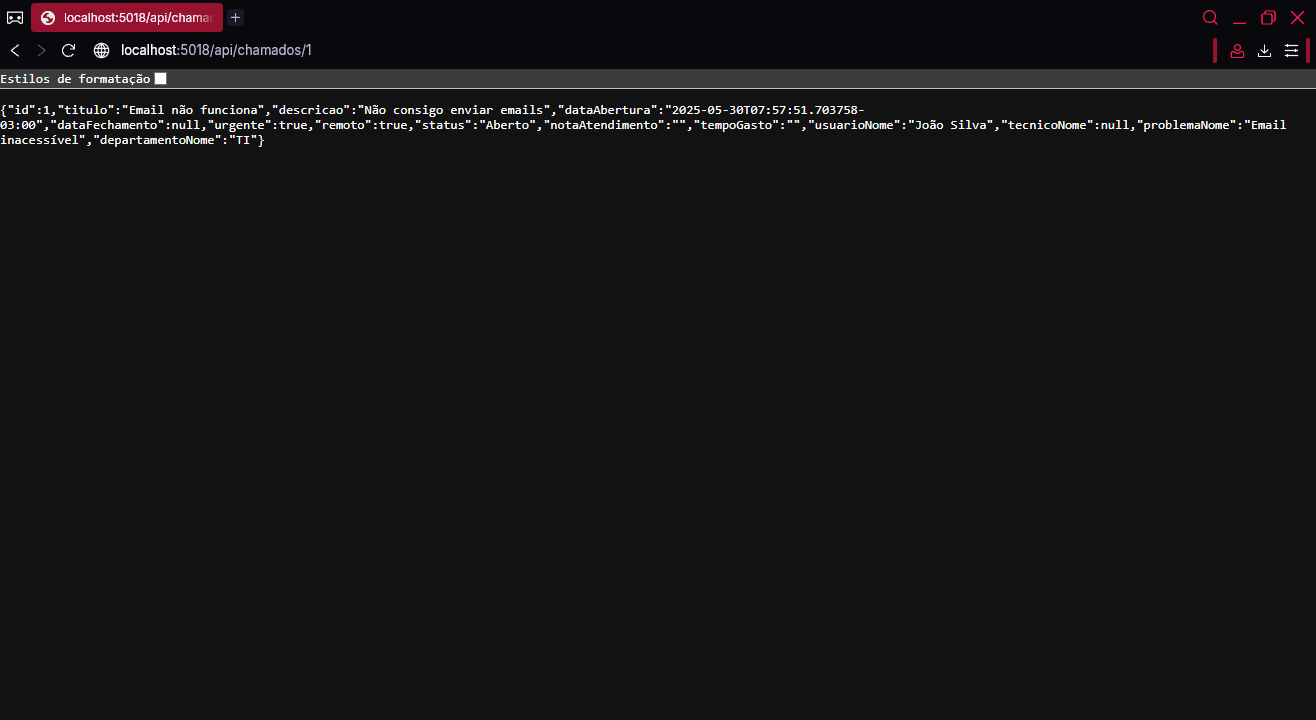
Fonte: Autoral

Figura 3 – LocalHost com a rota Get e com estilos de formatação ativo

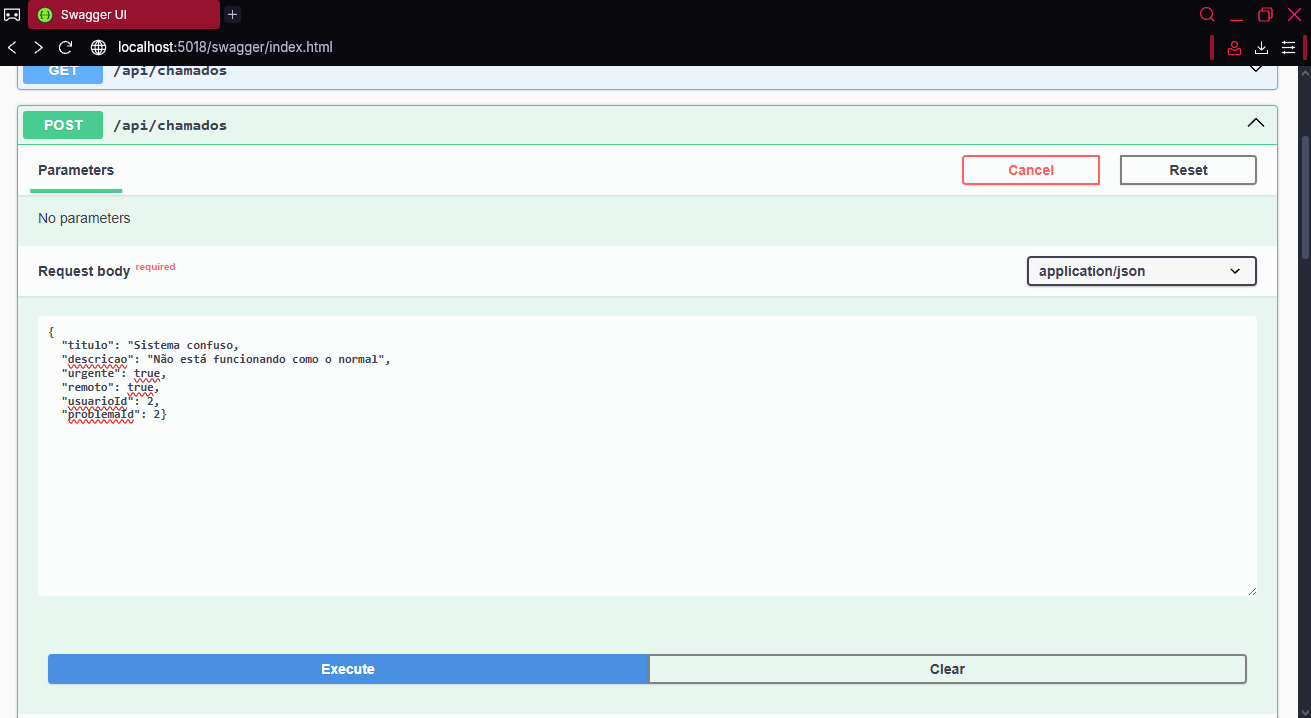


Fonte: Autoral

Figura 4 – LocalHost com a rota Get e um id

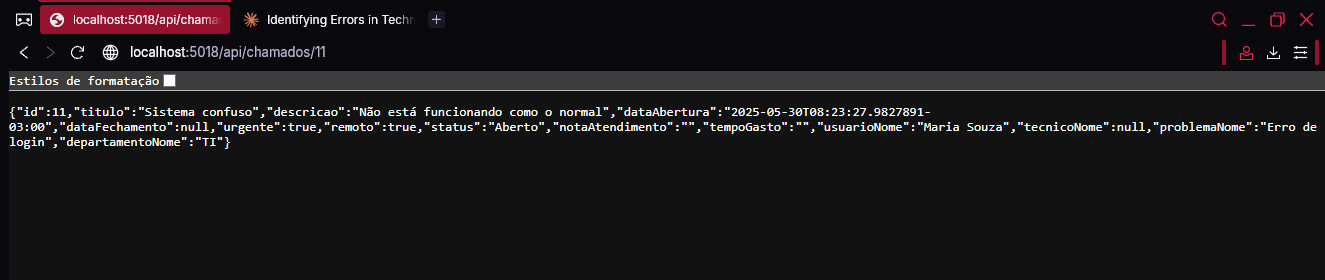


Fonte: Autoral

Figura 5 – Uso da rota Post (foi usado o auxílio do Swagger) 

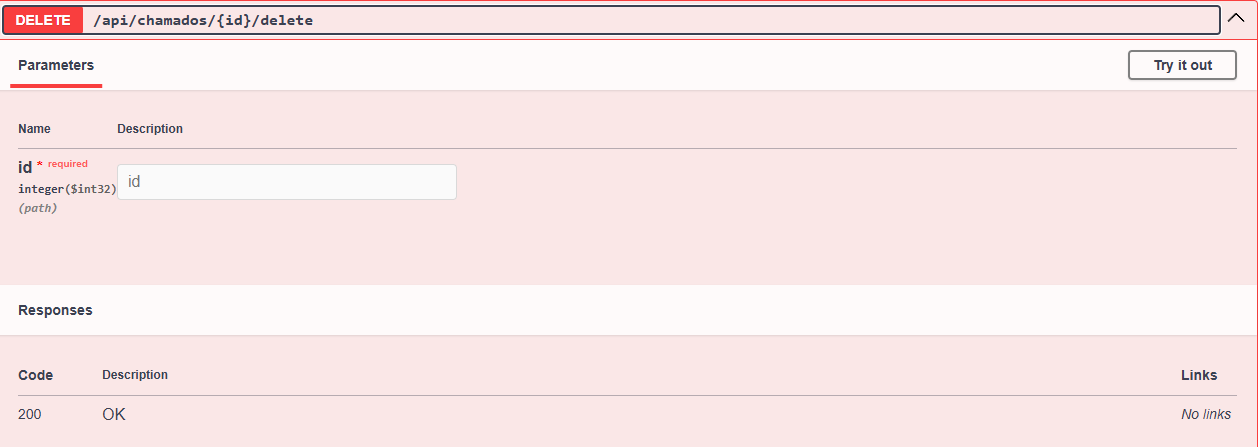
Fonte: Swagger UI

Figura 6 – Uso da rota Get após o Post

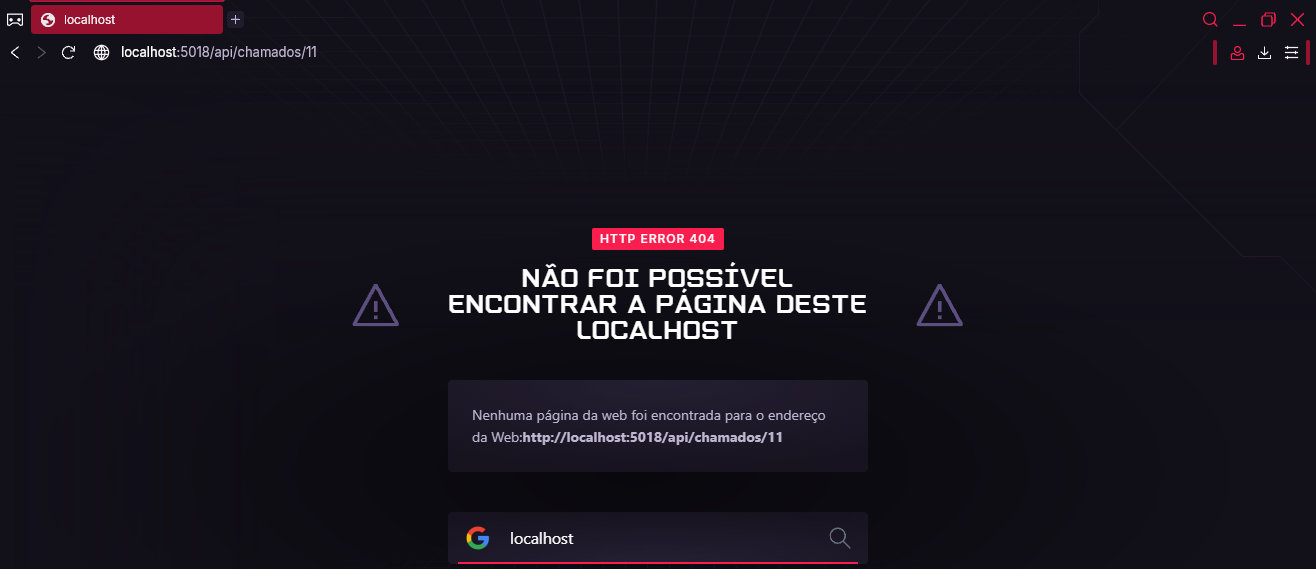


Fonte: Autoral

Figura 7 – Uso da rota Delete (foi usado o auxílio do Swagger)



Fonte: Swagger UI

Figura 8 – Uso da rota Get após o uso do Delete

Fonte: Autoral