

Sistema de Gestão para Clínica de Podologia

SQL Server • C# (.NET) API • React

Apresentação: arquitetura, banco, API e front-end — implementações e trechos de código.



Visão Geral & Arquitetura

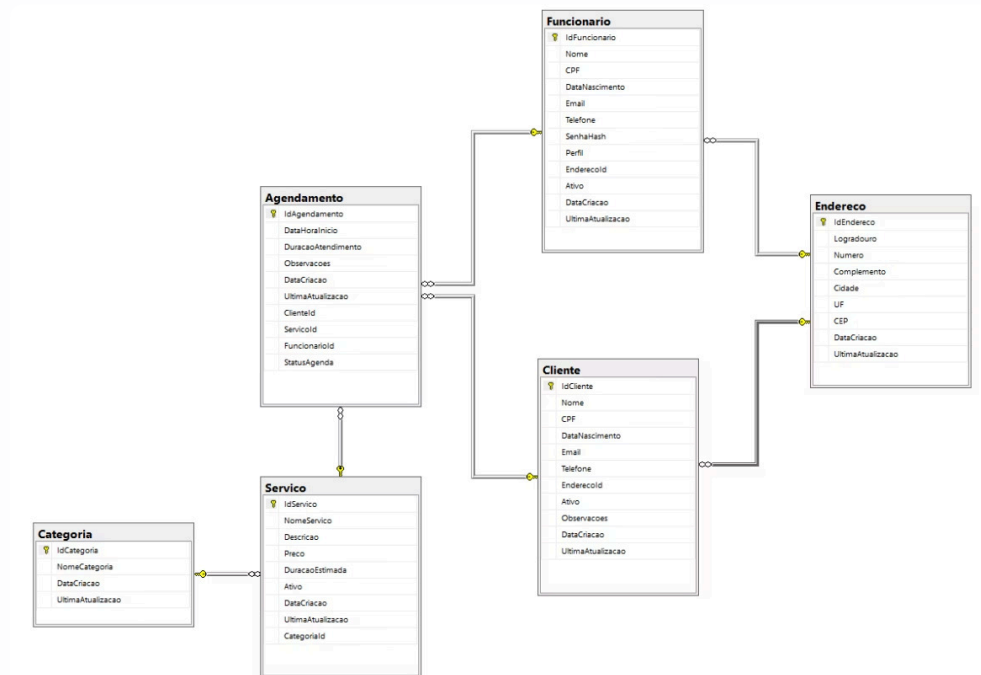
Objetivo: gerenciar agenda, clientes, serviços e relatórios com segurança e auditabilidade. Fluxo: React (UI) → API (.NET 9) → SQL Server (persistência). Autenticação via JWT, CORS controlado na API.

```
// Program.cs (trecho)
builder.Services.AddControllers();
builder.Services.AddCors(options =>
{options.AddPolicy("PermitirTudo", policy =>
    { policy.AllowAnyOrigin()
        .AllowAnyMethod()
        .AllowAnyHeader();});
});
//jwt
app.UseAuthentication();
app.UseAuthorization();
```

Modelo de Dados — Entidades Principais

Entidades centrais: Cliente (CPF), Funcionário (CPF), Serviço (duração, preço), Agendamento (data/hora, status), Endereço (FK Cliente). Chaves primárias e estrangeiras garantem integridade referencial.

```
-- CREATE TABLE Agendamento (trecho)
CREATE TABLE [dbo].[Agendamento](
    [IdAgendamento] [int] IDENTITY(1,1) NOT NULL,
    [DataHoraInicio] [datetime] NOT NULL,
    [DuracaoAtendimento] [int] NOT NULL,
    [Observacoes] [nvarchar](max) NULL,
    [DataCriacao] [datetime] NOT NULL,
    [UltimaAtualizacao] [datetime] NOT NULL,
    [Clientelid] [int] NOT NULL,
    [Servicoid] [int] NOT NULL,
    [Funcionarioid] [int] NOT NULL,
    [StatusAgendaId] [int] NOT NULL,
    PRIMARY KEY CLUSTERED
(
    [IdAgendamento] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE =
OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```





Estrutura & Scripts SQL

Entregáveis iniciais: scripts de criação de tabelas, seed de serviços e usuário admin, índices para consultas por data e CPF.

```
INSERT INTO Servico (Nome, DuracaoMin, Preco) VALUES ('Consulta Inicial', 45, 120.00);  
INSERT INTO Usuario (Login, SenhaHash, Role) VALUES ('admin', '...hashed...', 'Admin');  
CREATE INDEX IX_Agend_DataHora ON Agendamento(DataHora);
```

API: Configuração e Entidades

Stack: .NET 9, ASP.NET Core MVC, EF Core, Repository pattern e Swagger para documentação. Separação clara: Controllers → Services → Repositories → DTOs.

```
// Cliente.cs (modelo)
public class Cliente : Pessoa
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int IdCliente { get; set; }
    public string Observacoes { get; set; }
    public bool Ativo { get; set; }

    public int? Enderecoid { get; set; }
    public Endereco Endereco { get; set; }

    Agendamentos { get; set; } = new
    List<Agendamento>();

    public Cliente()
    {
    }

    public Cliente(int idCliente)
    {
        IdCliente = idCliente;
    }
}
```

```
// ClienteDto.cs
namespace GestaoClinica.DTO;

public class ClienteDTO : PessoaDTO
{
    public int IdCliente { get; set; }
    public string Observacoes { get; set; }
    public bool Ativo { get; set; }
    public int? Enderecoid { get; set; }
    public EnderecoDTO? Endereco { get; set; }
}

public class ClienteResumoDTO : PessoaResumoDTO
{
    public int IdCliente { get; set; }
}
```

API: Funcionalidades e Endpoints

Endpoints completos para CRUD de Clientes, Funcionários, Serviços e Agendamentos. Autenticação via JWT, roles e policies para permissões (Admin, Recepção, Podólogo).

```
// Controllers/ClientesController.cs
using Microsoft.AspNetCore.Mvc;
using GestaoClinica.Services.Interfaces;
using GestaoClinica.Entities;
using System.Net.Mime;

namespace GestaoClinica.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Produces(MediaTypeNames.Application.Json)]
    public class ClientesController : ControllerBase
    {
        private readonly IClienteService _clienteService;

        public ClientesController(IClienteService clienteService)
        {
            _clienteService = clienteService;
        }

        [HttpGet]
        public async Task<ActionResult> GetClientes()
        {
            try
            {
                var clientes = await _clienteService.ListarClienteAsync();

                var resultado = clientes.Select(c => new
                {
                    c.IdCliente,
                    c.Observacoes,
                    c.Ativo,
                    c.Nome,
                    c.Telefone,
                    c.Email,
                    CPF = c.CPF,
                    c.DataNascimento,
                });
            }
        }
    }
}
```

```
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteCategoria(int id)
{
    try
    {
        await _clienteService.ExcluirAsync(id);
        return Ok(new { message = $"Cliente com ID {id}
excluído com sucesso!" });
    }
    catch (KeyNotFoundException)
    {
        return NotFound(new { message = $"Cliente com ID
{id} não encontrado." });
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "Erro ao
excluir cliente.", error = ex.Message });
    }
}
```

Front-end: Stack e Rotas

Stack: React, styled-components, axios, react-router.

Organização: /src/pages, /src/components. Rotas públicas (login) e privadas (dashboard, agenda) com verificação de token no contexto.

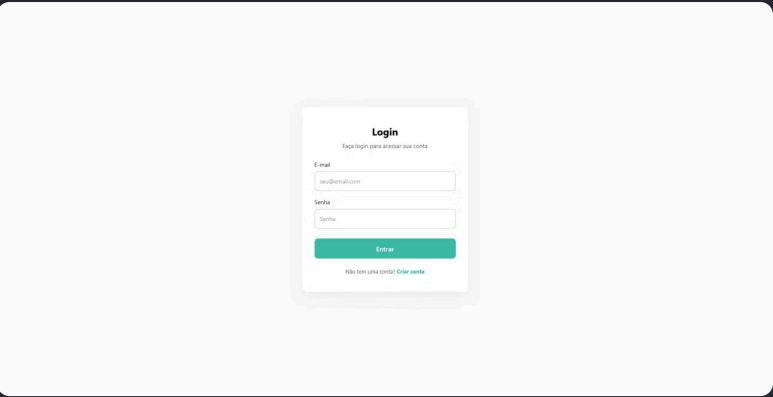
```
try {  
  const response = await  
  axios.get(`http://localhost:${REACT_APP_PORT}/api/client  
es`);
```

```
  const toList = (payload) =>  
    Array.isArray(payload?.data)  
      ? payload.data  
      : Array.isArray(payload)  
        ? payload  
        : (payload?.data?.$values ?? payload?.$values ?? []);
```

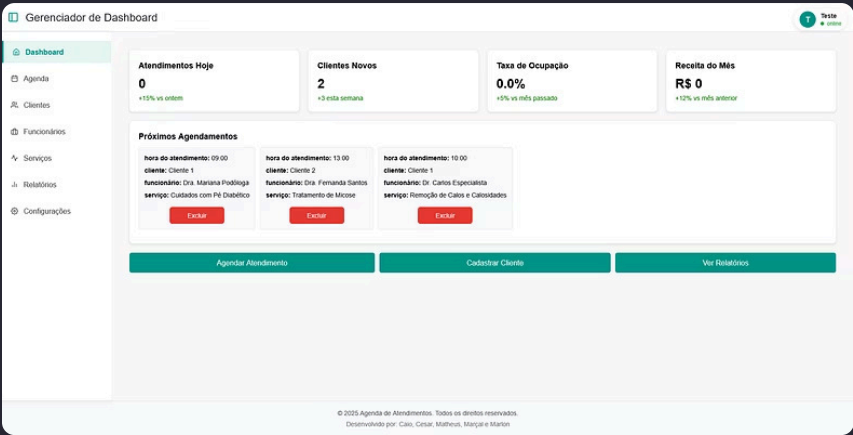
```
  setClientes(toList(response.data));  
} catch (error) {  
  console.error("Erro ao buscar clientes:", error);  
}  
};
```

| Cliente | Contato | Status | Ações |
|-----------|---------------------------------|--------|-------------------|
| Cliente 1 | 6195555555 teste@teste1.com | Ativo | Atualizar Excluir |
| Cliente 2 | 45621312645 teste@teste2.com | Ativo | Atualizar Excluir |

Login e Dashboard



Login salva token no localStorage com refresh token opcional. Rotas protegidas dependem do token e claims.



Dashboard com KPIs: atendimentos do dia, faturamento, taxa de ocupação, clientes novos. Chamadas agregadas à API.

```
// useDashboard.js (trecho)
useEffect(() => {
  const fetchDados = async () => {
    const [agendamentos, clientes, servicos, funcionarios] = await
Promise.all([

fetch(`http://localhost:${REACT_APP_PORT}/api/agendamentos`).t
hen(res => res.json()).catch(() => ({})),

fetch(`http://localhost:${REACT_APP_PORT}/api/clientes`).then(res
=> res.json()).catch(() => ({})),

fetch(`http://localhost:${REACT_APP_PORT}/api/servicos`).then(re
s => res.json()).catch(() => ({})),

fetch(`http://localhost:${REACT_APP_PORT}/api/funcionarios`).the
n(res => res.json()).catch(() => ({})),

]);
    fetchDados();
  }, [atualizarDados]);
```


Agenda e Clientes

Agenda: criar, editar, cancelar com validação de conflitos por funcionário e serviço. Clientes: CRUD, busca por nome/CPF e paginação server-side para tabelas grandes.

```
// Carregar dados agendamento
const fetchDados = async () => {
  try {
    const [agendasRes, servicosRes, funcionariosRes] = await
      Promise.all([

        fetch(`http://localhost:${REACT_APP_PORT}/api/agendamentos`)
          .then(r => r.json()).catch(() => ({})),

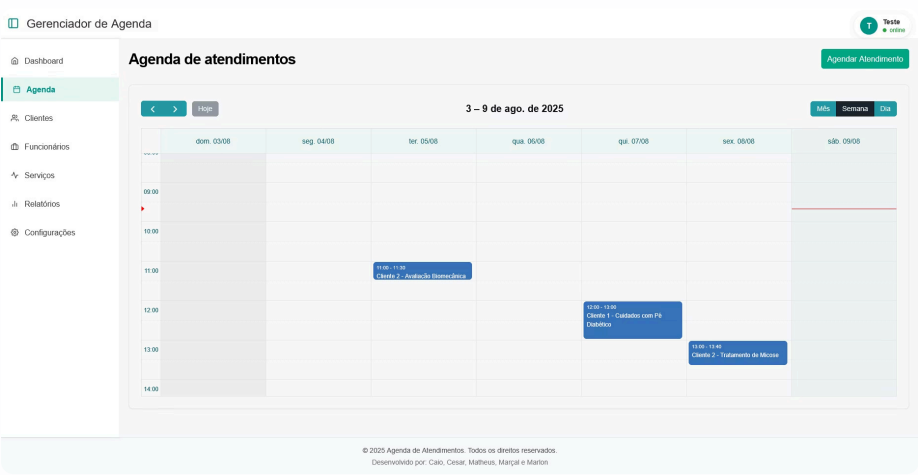
        fetch(`http://localhost:${REACT_APP_PORT}/api/servicos`)
          .then(r => r.json()).catch(() => ({})),

        fetch(`http://localhost:${REACT_APP_PORT}/api/funcionarios`)
          .then(r => r.json()).catch(() => ({})),

      ]);

    setAgendamentos(coalesceList(agendasRes));
    setServicos(coalesceList(servicosRes));
    setFuncionarios(coalesceList(funcionariosRes));
  } catch (e) {
    console.error(e);
    setAgendamentos([]);
  }
};

fetchDados();
```





Funcionários, Serviços e Relatórios

Funcionários com perfis e permissões. Serviços com duração e preço padrão para cálculo automático de agenda. Relatórios: produção por profissional.

//html do relatório

```
<div className="card-title">Receita Total</div>
  <div className="kpi">
    <span className="label">Total</span>
    <span className="value">R$ {relatorio.receita.toLocaleString('pt-BR', { minimumFractionDigits: 2,
maximumFractionDigits: 2 })}</span>
  </div>
```