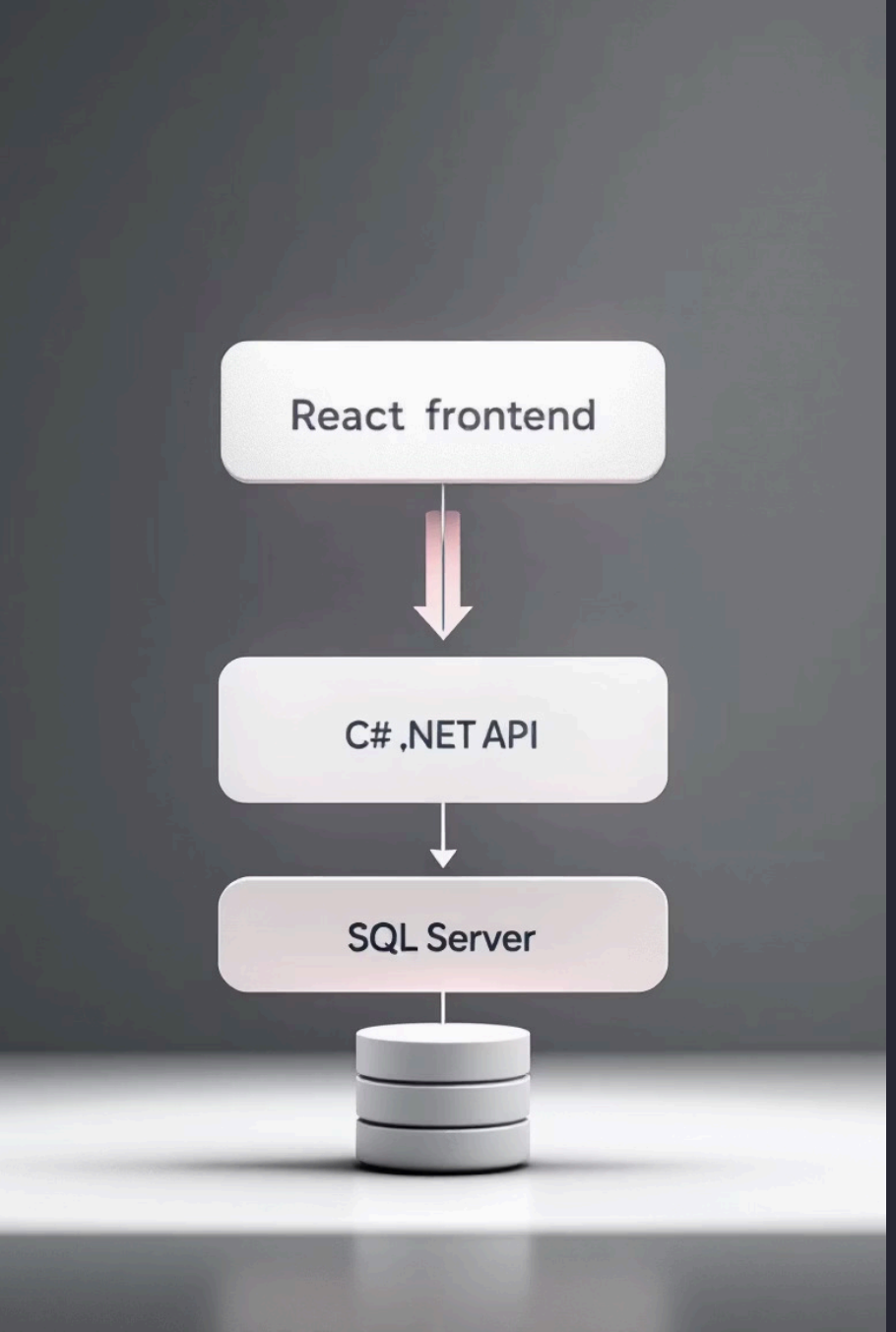
Sistema de Gestão para Clínica de Podologia

SQL Server • C# (.NET) API • React

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



Visão Geral & Arquitetura

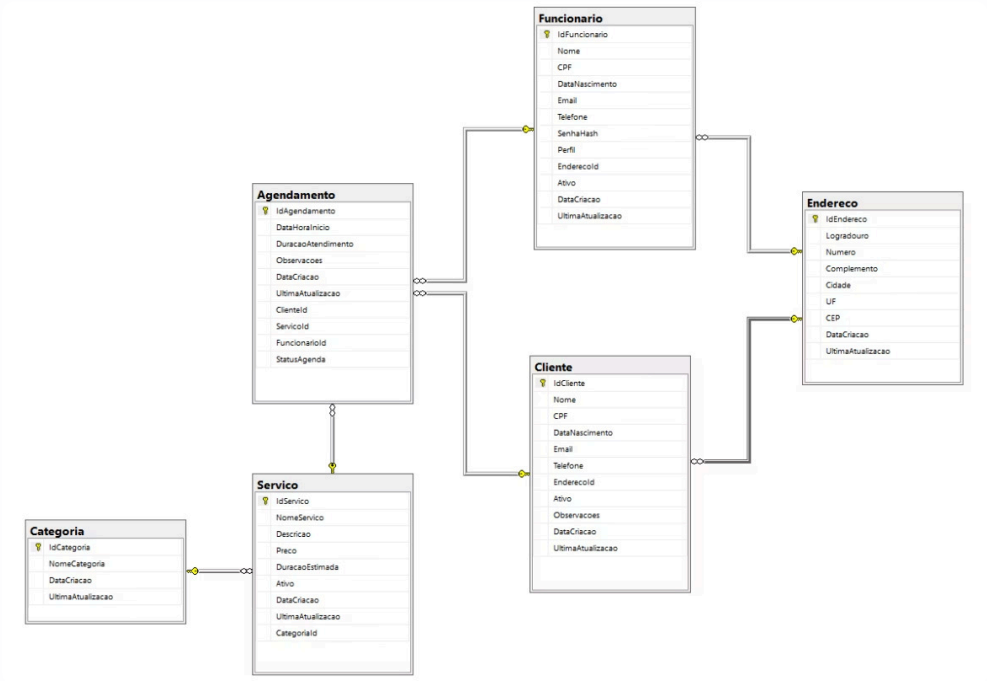
Objetivo: gerenciar agenda, clientes, serviços e relatórios com segurança e auditabilidade. Fluxo: React (UI) → API (.NET 9) → SQL Server (persistência). Autenticação via JWT, CORS controlado na API.

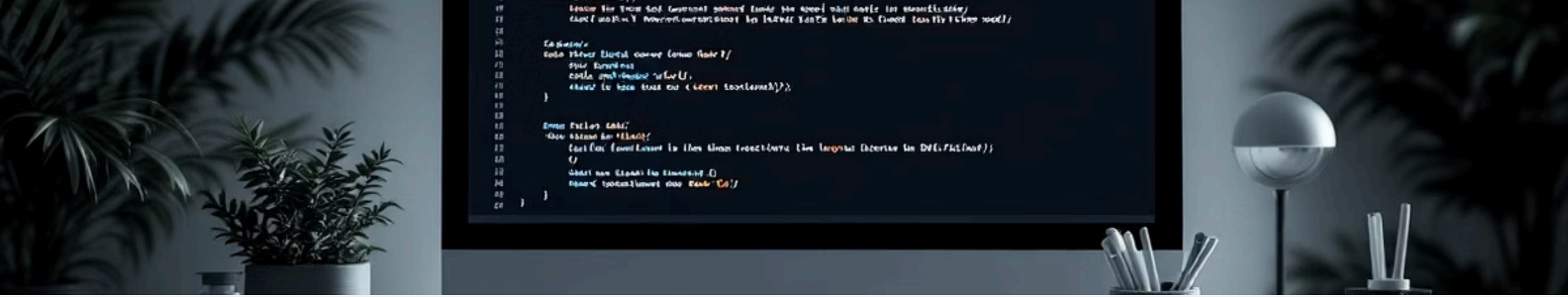
```
// Program.cs (trecho)
builder.Services.AddControllers();
builder.Services.AddCors(options =>
{options.AddPolicy("PermitirTudo", policy =>
   { policy.AllowAnyOrigin()
           .AllowAnyMethod()
           .AllowAnyHeader();});
});
//jwt
app.UseAuthentication();
app.UseAuthorization();
```

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,
	[ClienteId] [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

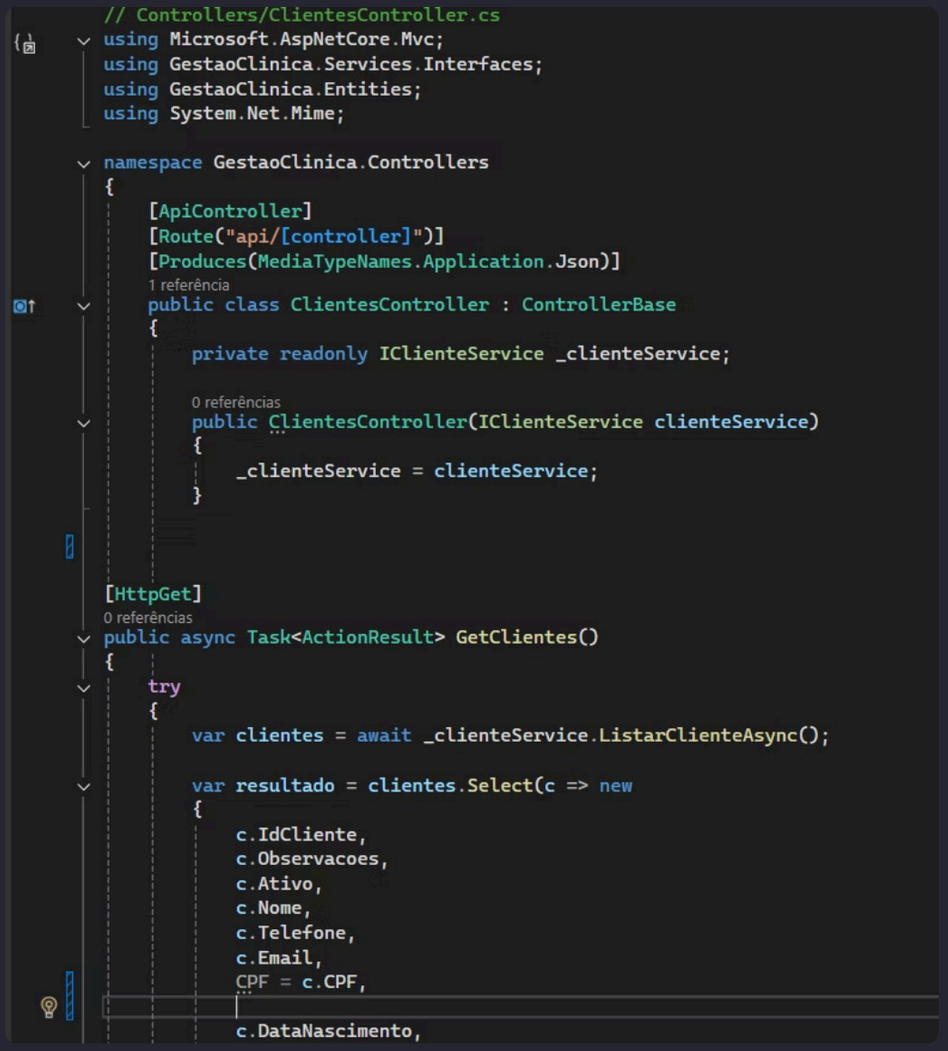
Stack: .NET 9, ASP.NET Core MVC, EF Core, Repository pattern e Swagger para documentação. Separação clara: Controllers → Services → Repositories → DTOs.

```
       // Cliente.cs (modelo)
 public class Cliente : Pessoa
 {
    [Key]
[DatabaseGenerated(DatabaseGeneratedOption.Ide
ntity)]
    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).



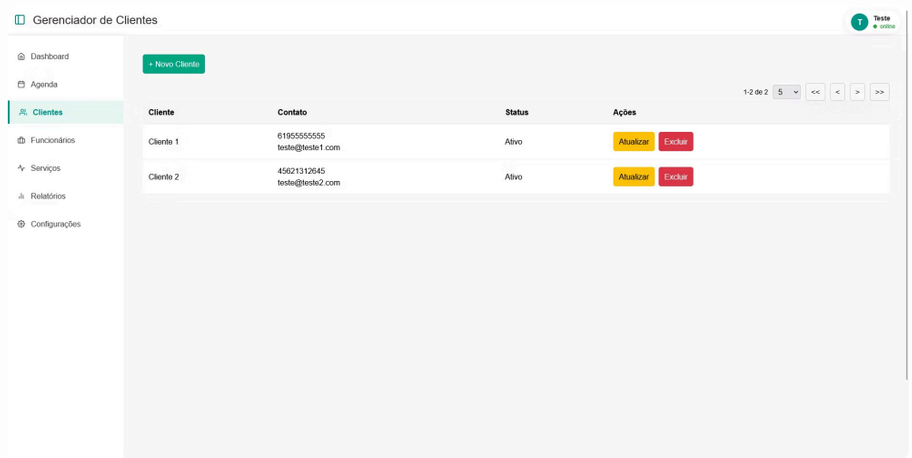
```
// Controllers/ClientesController.cs
using Microsoft.AspNetCore.Mvc;
using GestaoClinica.Services.Interfaces;
using GestaoClinica.Entities;
using System.Net.Mime;

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

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


[HttpGet]
0 referências
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<IActionResult> 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 });
    }
}
```

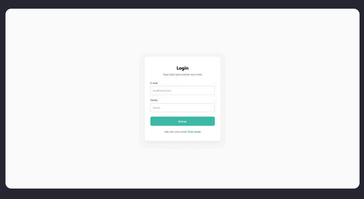
Front-end: Stack e Rotas

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.

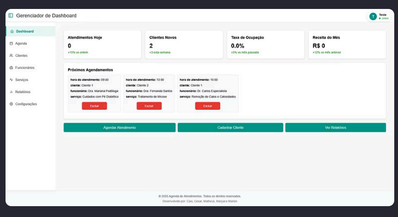
```
 try {
     const response = await
axios.get(`http://localhost:${REACT_APP_PORT}/api/client
es`);



    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);
  }
 };
```



Login e Dashboard



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

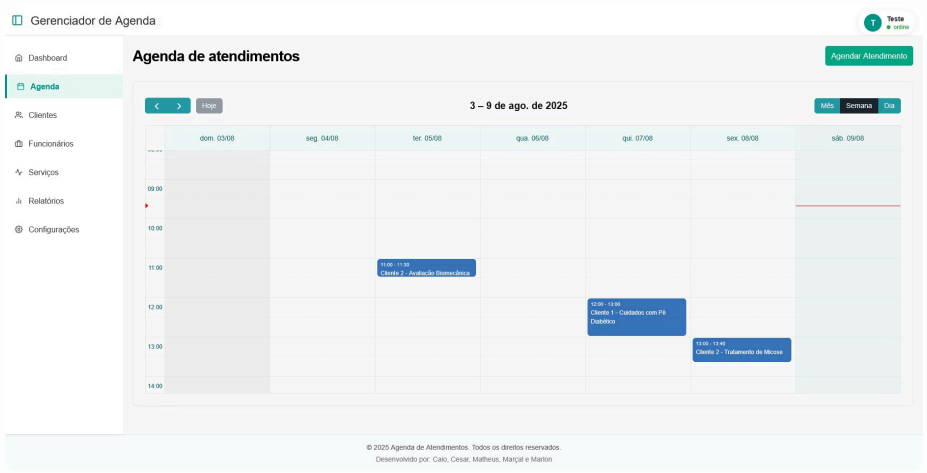


Dashboard com KPIs: atendimentos do dia, faturamento, taxa de ocupação, clientes novos. Chamadas agregadas à API.

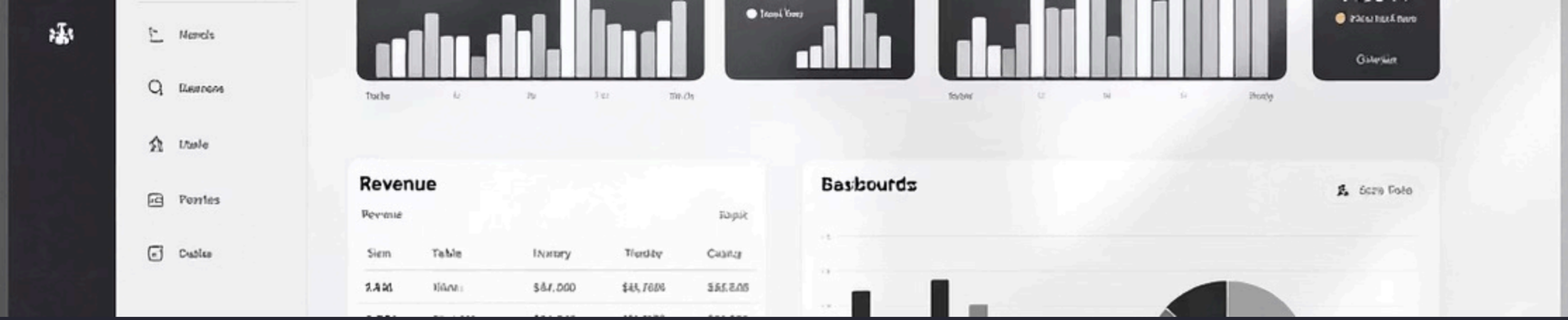
```
// useDashboard.js (trecho)
useEffect(() => {
  const fetchDados = async () => {
    const [agendamentos, clientes, servicos, funcionarios] = await Promise.all([

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

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

    const listaAg = Array.isArray(agendamentos?.data)
      ? agendamentos.data
      : (agendamentos?.data?.$values ?? []);

    const listaClientes = Array.isArray(clientes?.data)
      ? clientes.data
      : (clientes?.data?.$values ?? []);

    const listaFuncs = Array.isArray(funcionarios?.data)
      ? funcionarios.data
      : (funcionarios?.data?.$values ?? []);

    // Cria mapa idFuncionario -> nome
    const mapa = {};
    for (const f of (listaFuncs ?? [])) {
      const id = f?.idFuncionario ?? f?.id ?? f?.funcionarioId;
      const nome = f?.nome ?? f?.name ?? "Sem nome";
      if (id != null) mapa[id] = nome;
    }
    setFuncMap(mapa);

    const hoje = new Date().toDateString();

    const atendimentosHoje = (listaAg ?? []).filter(
      a => new Date(a.dataHoraInicio).toDateString() === hoje
    ).length;

    const totalHorariosDisponiveisHoje = 8;
    const taxaOcupacao = ((atendimentosHoje / totalHorariosDisponiveisHoje) * 100).toFixed(1);

    const clientesNovos = (listaClientes ?? []).filter(c => {
      const dc = c?.dataCriacao ? new Date(c.dataCriacao) : null;
      const seteDiasAtras = new Date(new Date().setDate(new Date().getDate() - 7));
      return dc && dc >= seteDiasAtras;
    }).length;

    const receitaMes = (listaAg ?? []).reduce((acc, ag) => acc + (ag.servico?.preco || 0), 0);

    // Ordena agendamentos por data/hora e pega os próximos 5
    const proximos = [...(listaAg ?? [])]
      .sort((a, b) => new Date(a.dataHoraInicio) - new Date(b.dataHoraInicio))
      .slice(0, 5);

    setDados({
      atendimentosHoje,
      clientesNovos,
      taxaOcupacao,
      receitaMes,
      proximos,
    });
  };

  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`).th
en(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 relatorio
<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>
```