Tech SAÚDE

Projeto de Sistemas

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

.

Discentes
Alicia da Silva Rodrigues
Cibelly Angel da Silva
Grasielly Ribeiro da Silva
Renata da Silva Camargo
Sophia Freire Bezerra

Professor Orientador
Mauricio Asenjo Neves

Curso
Técnico em Informática
CTII 448

Cubatão
2024

Resumo

O presente trabalho tem como objetivo a execução do conhecimento interdisciplinar do curso Técnico em Informática, ofertado pelo Instituto Federal de Educação, Ciência e Tecnologia do Estado de São Paulo — Campus Cubatão, através de um sistema de armazenamento de dados voltado ao histórico médico de pacientes. Desse modo, ofertamos uma ferramenta de cunho social visando o melhoramento do atendimento ao paciente por meio de coleta e armazenamento de informações médicas obtidas em eventuais consultas e exames. Para tanto, recorremos à metodologia de pesquisa qualitativa oferecendo três possibilidades de se realizar tal atividade: a pesquisa documental, o estudo de caso e também, a etnografia — importante para compreender a realidade dos usuários da área da saúde, suas principais necessidades e apontamentos de melhoria para o sistema de saúde brasileiro.

Palavras-chaves: Armazenamento de dados. Segurança da Informação. Saúde. Lei Geral de Proteção de Dados.

Introdução

O sistema de armazenamento de dados médicos, Tech Saúde, é uma ferramenta de cunho social que visa o melhoramento do atendimento ao paciente, propondo um meio seguro de coleta e armazenagem de informações médicas geradas em consultas rotineiras. Para isso, fundamenta-se nos princípios da Lei Geral de Proteção de Dados (LGPD) e nas noções do Ministério da Saúde (MS) a fim de proporcionar uma aplicação web com as devidas atenções às necessidades dos pacientes, oferecendo-lhes um ambiente digital com instrumentos para facilitar o acesso aos dados gerados em consultas médicas, como, por exemplo, informações da anamnese, indexação de documentos e agendamentos.

O projeto de sistemas referido é uma aplicação web desenvolvida com o Asp.Net e React, sendo o primeiro utilizado para a criação da web API no Back-end com a linguagem de programação C#, enquanto o segundo tem como foco o Front-end em JavaScript.

Sumário

Resumo **3**
Introdução **4**
Sumário **5**
Ambiente de desenvolvimento integrado (IDE) **7**
IDES utilizadas no projeto **7**
Visual Studio Code 7
Visual Studio Community 2022 12
Front-End **15**
Linguagem de programação 15
JavaScript 15
Framework 15
React 15
Bibliotecas 16
Bootstrap 16
Axios 16
Código de Validações 16
Páginas de navegação 16
Administrador 17
Paciente 20
Perfil 21
Histórico 22
Consultas 24
Agendadas 26
Histórico 26
Cadastro 29
Back-End **35**
Linguagem de programação 35
C# 35
Framework 35
Asp.Net 35
Bibliotecas 35
Entity Framework (EF) 35
JSON Web Tokens (JWT) 36
Conceitos 36
Código 36
Conexão com o banco de dados 37
Modelos 37
Controllers 38
Repository Pattern 38
Apresentação de telas **63**

Conclusão **69**
Referências **70**
ANEXO I - Back-end: **71**
ANEXO II - Front-end: **71**
Front-end: 71
Código da página do CRUD: 71
Código da página do ADM: 72
Código da página de Perfil: 73
Back-end: 74
Anexo 1 - BackEnd: Instalação das bibliotecas 75
Anexo 2 - BackEnd : Código de Conexão com o Banco de Dados 76
Anexo 3 - BackEnd : Código do Program.cs 77
Anexo 4 - BackEnd : Códigos dos Modelos 79
Anexo 5 - BackEnd : Código dos Controllers 89
Anexo 6 - BackEnd : Código do Repositório das Alergias 96
Anexo 7 - BackEnd : Código do Repositório Auth 98
Anexo 8 - BackEnd : Código do Repositório Consultas 100
Anexo 9 - BackEnd : Repositório de Documentos 104
Anexo 10 - BackEnd : 105

Ambiente de desenvolvimento integrado (IDE)

O ambiente de desenvolvimento integrado (IDE) é um programa voltado para criação de software a partir de ferramentas gerais em uma única interface gráfica (GUI). Para isso, o IDE geralmente consiste em:

- Editor de código-fonte: é um editor de texto que auxilia na criação de software por meio de funcionalidades como destaque da sintaxe com indicadores visuais, recurso de preenchimento automático específico da linguagem e verificação de erros durante o desenvolvimento.
- Automação de compilação local: são utilitários que automatizam tarefas simples e repetíveis durante a criação de uma compilação local do software.
- Debugger: é um programa usado para testar outros programas e mostrar graficamente a localização dos erros no código original.

IDES utilizadas no projeto

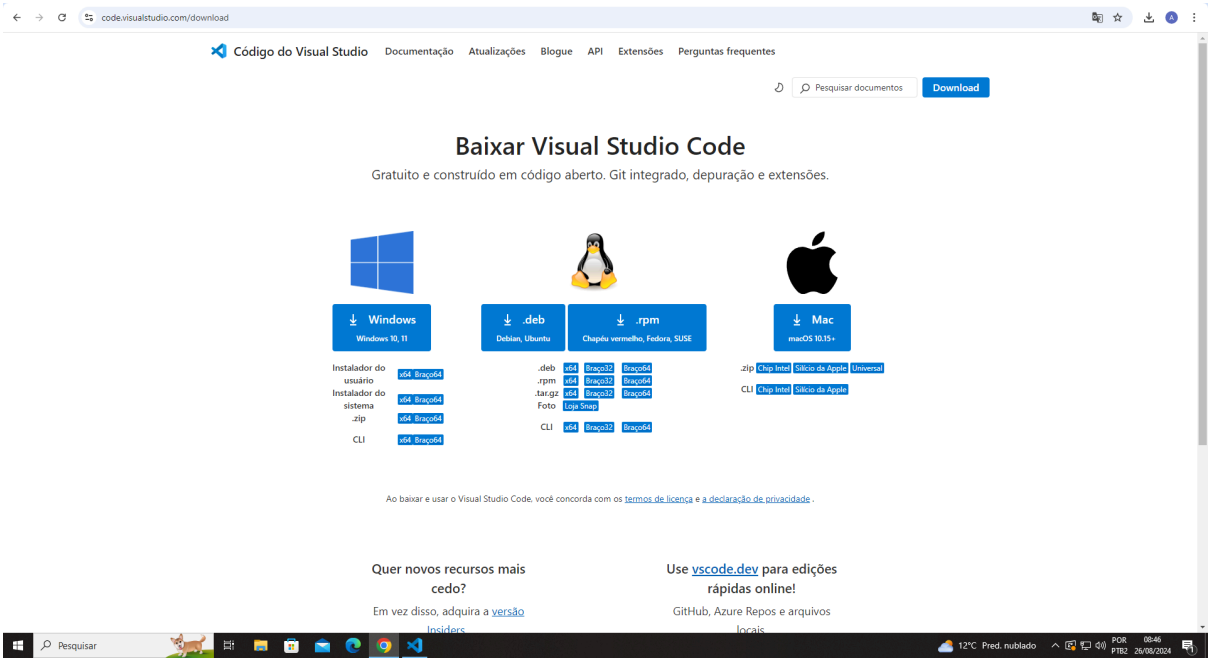
Para o desenvolvimento do projeto TechSaúde foram usadas as IDES: Visual Studio Code e Visual Studio Community 2022, sendo a primeira voltada ao Front-End e a segunda ao Back-End.

Visual Studio Code

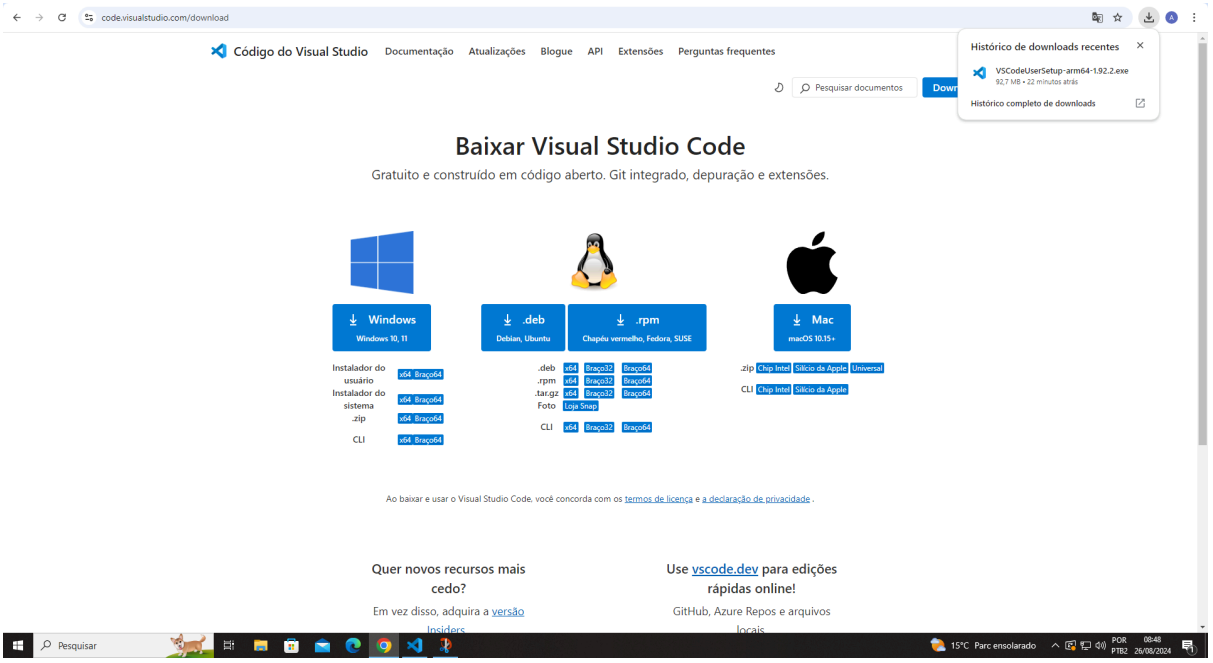
O Visual Studio Code (VSCode) é um editor de código-fonte desenvolvido pela Microsoft com o objetivo de desenvolver software e editar código. Fornece suporte interno para JavaScript e Node.js e com um ecossistema avançado de extensões para outras linguagens e runtimes, como por exemplo o C#.

Etapas de instalação

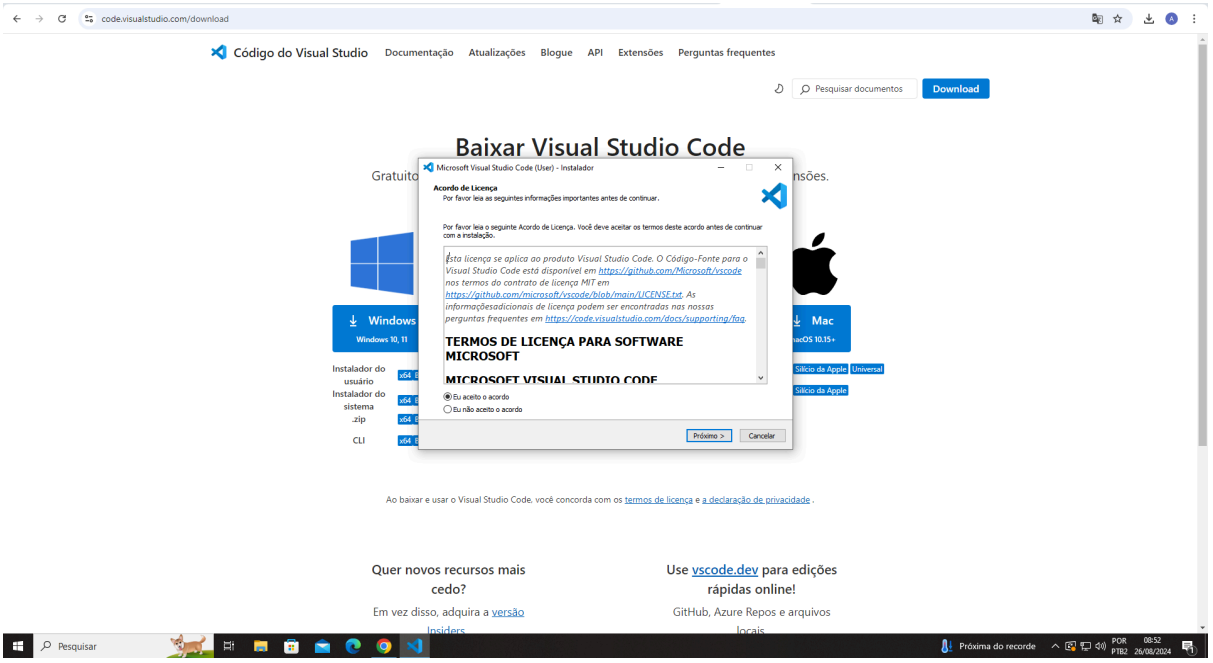
Para baixar o VsCode em sua máquina, primeiro acesse o site da Microsoft através do link: https://code.visualstudio.com/download e, depois, faça download da versão do Sistema Operacional (SO) correspondente.



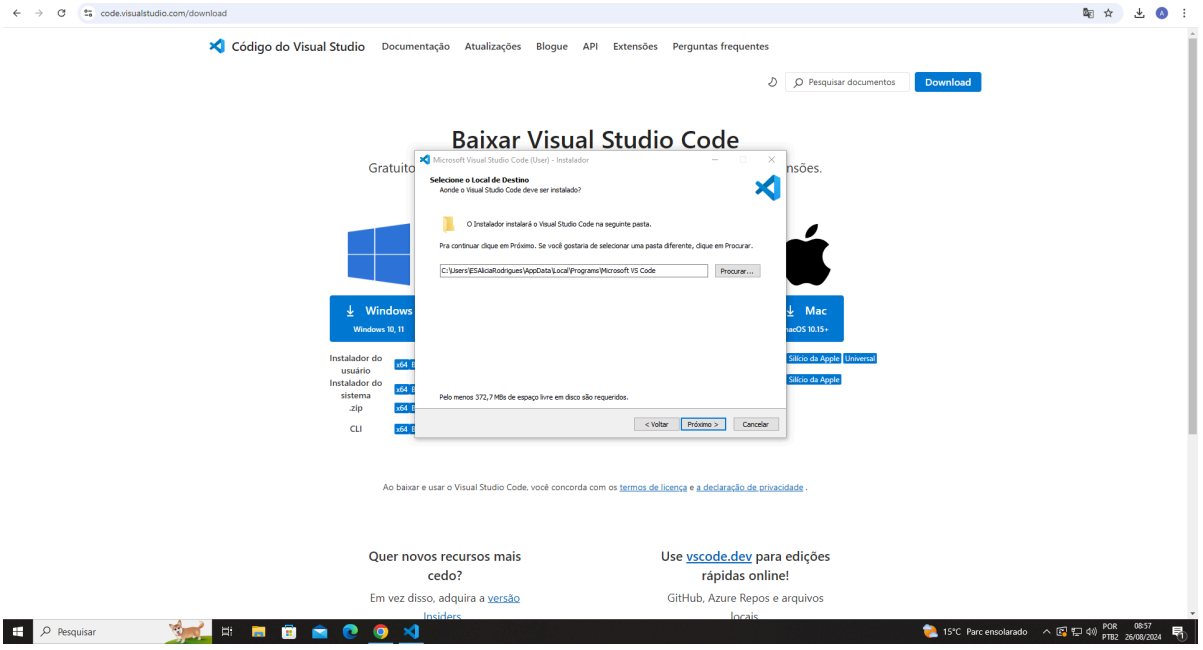
Após seguir as instruções iniciais, dê um duplo clique no download para continuar com a instalação da IDE.



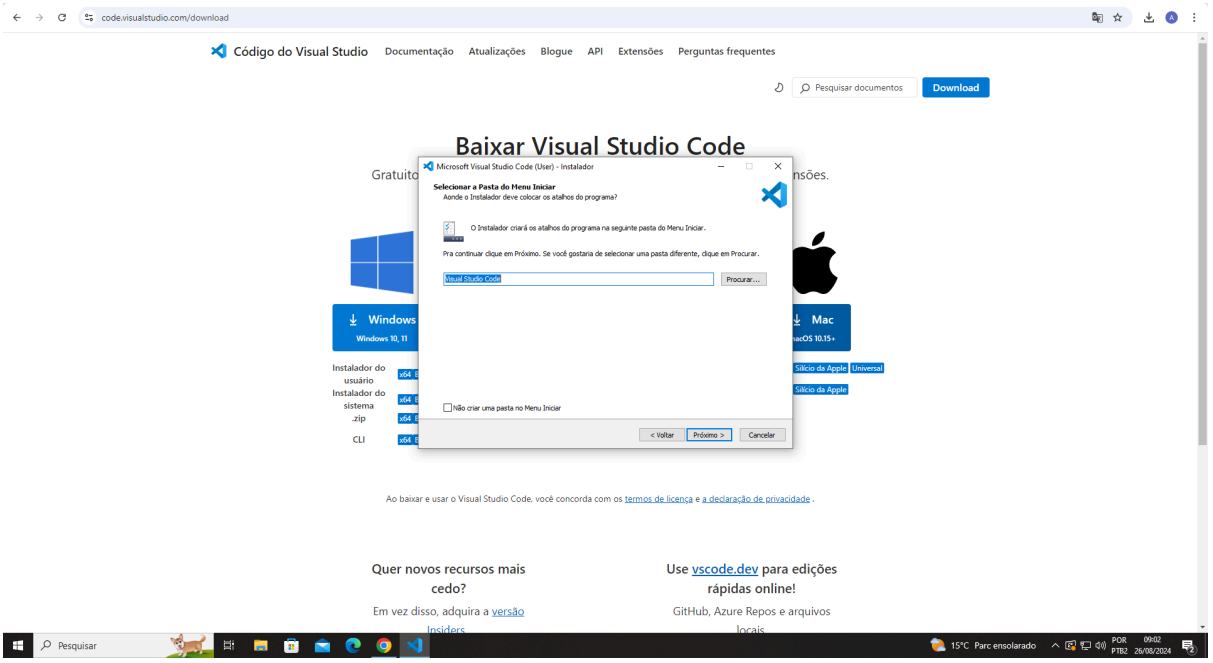
Feito isso, o VSCode será iniciado na tela principal de sua máquina. Leia e aceite os termos de licença para o uso do aplicativo para prosseguir com a instalação.



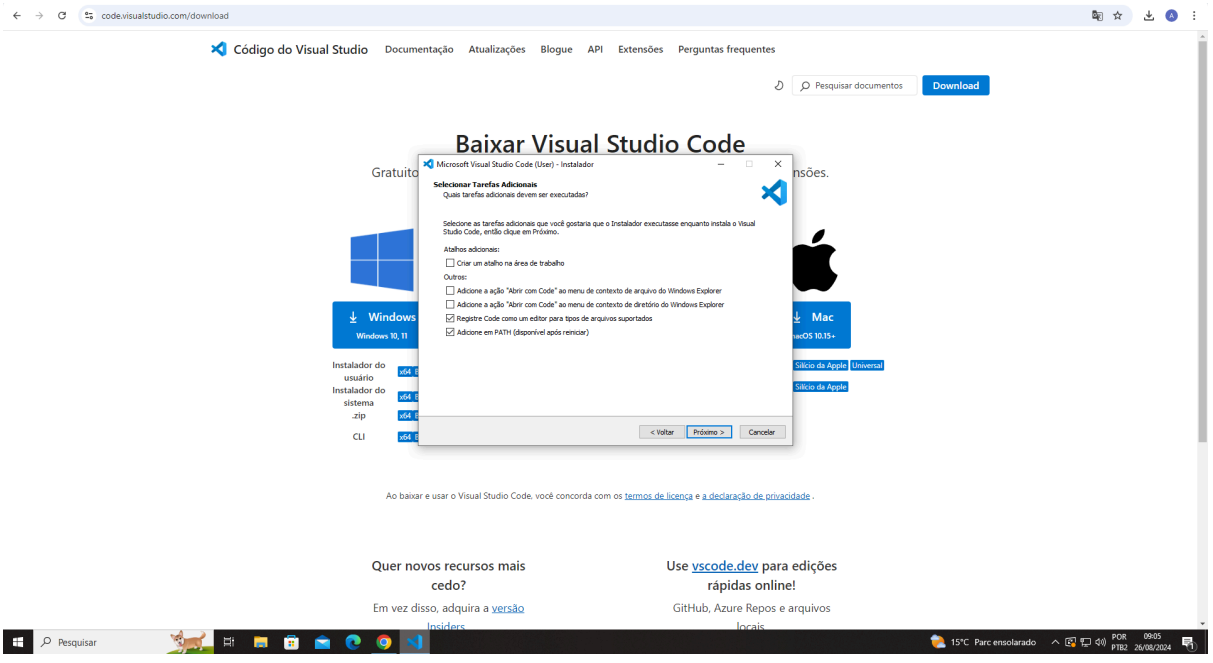
Agora, selecione o local de destino (pasta) em que o aplicativo deve ser instalado em sua conta do usuário:



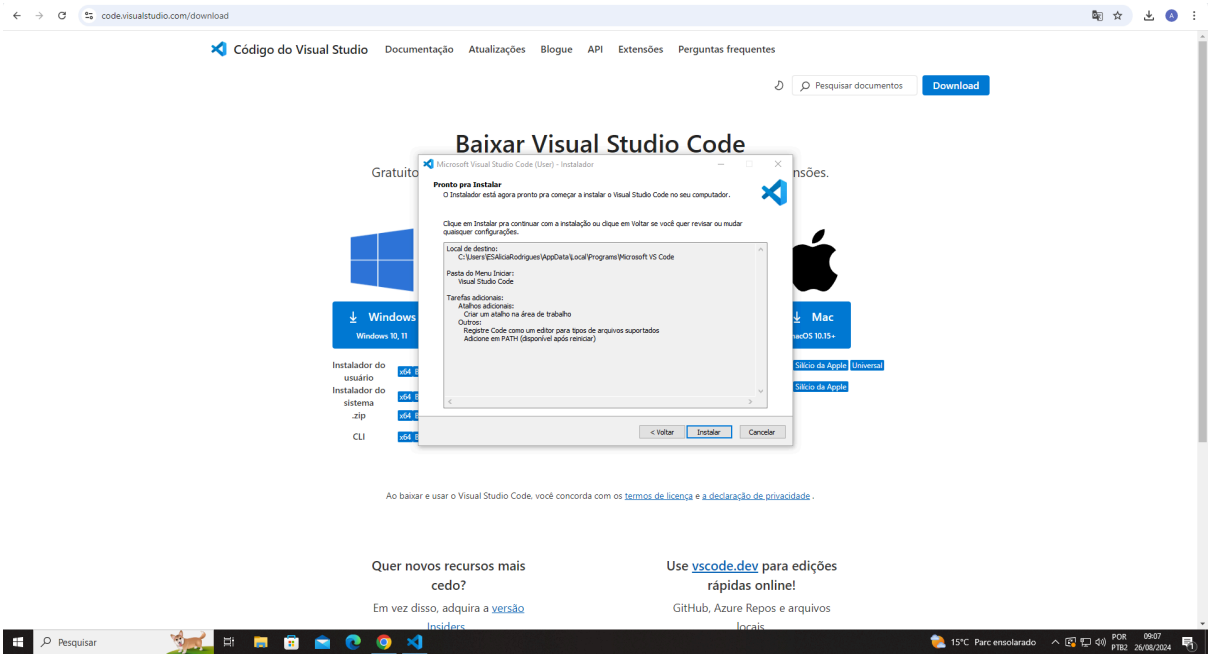
Selecione também a pasta do menu iniciar onde o Instalador deve colocar os atalhos do programa:



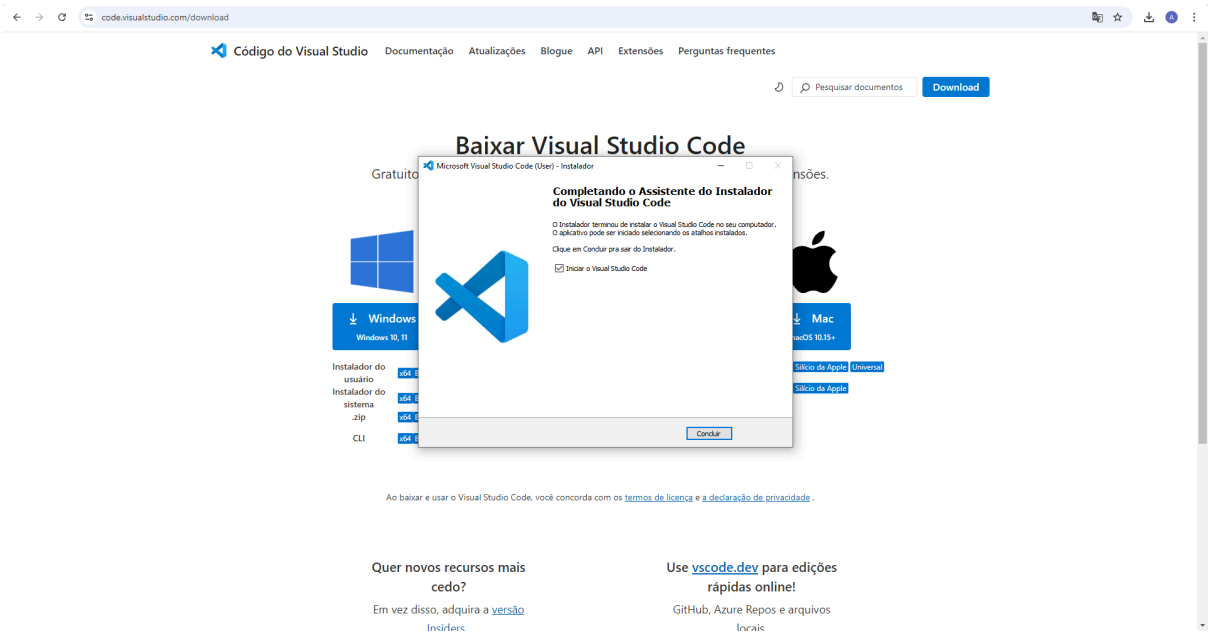
Caso deseje, selecione as tarefas adicionais que gostaria que o Instalador execute enquanto realiza a instalação do VSCode em sua máquina:



Pronto! Agora o Instalador tem todas as pré-configurações selecionadas e está pronto para iniciar a instalação em seu computador:



Assim que o Instalador extrair os arquivos em sua máquina, selecione o atalho desejado para iniciar o aplicativo e clique em concluir:



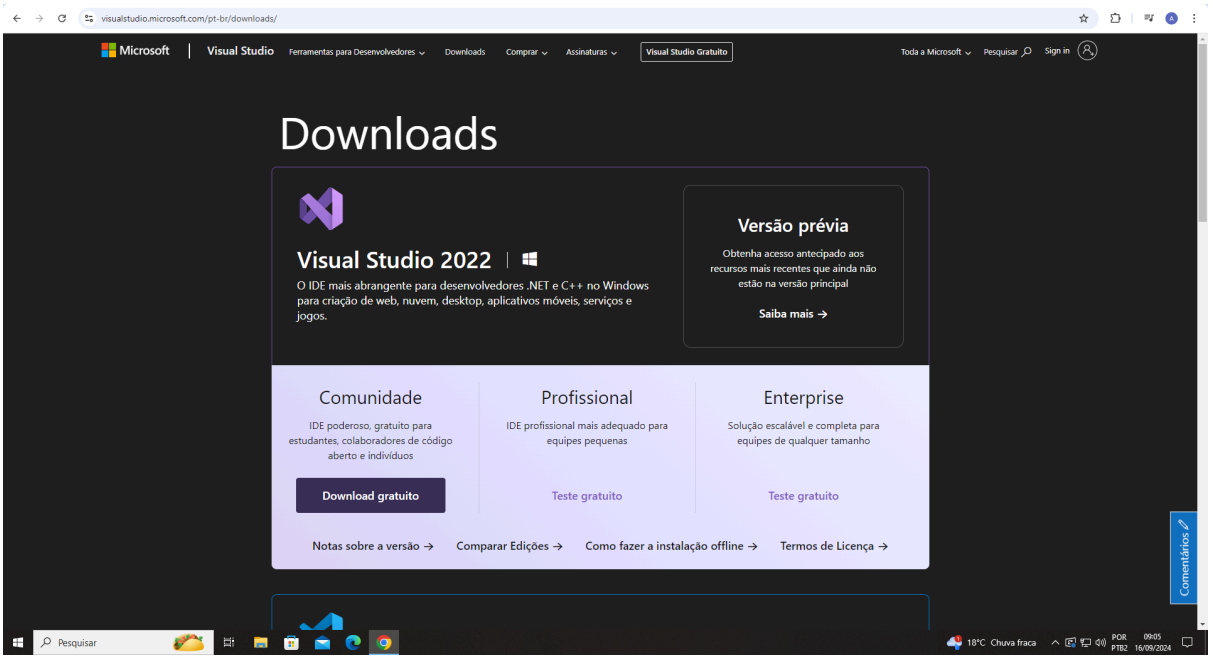
O Visual Studio Code está pronto para ser usado para o desenvolvimento de aplicações voltadas ao Front-End.

Visual Studio Community 2022

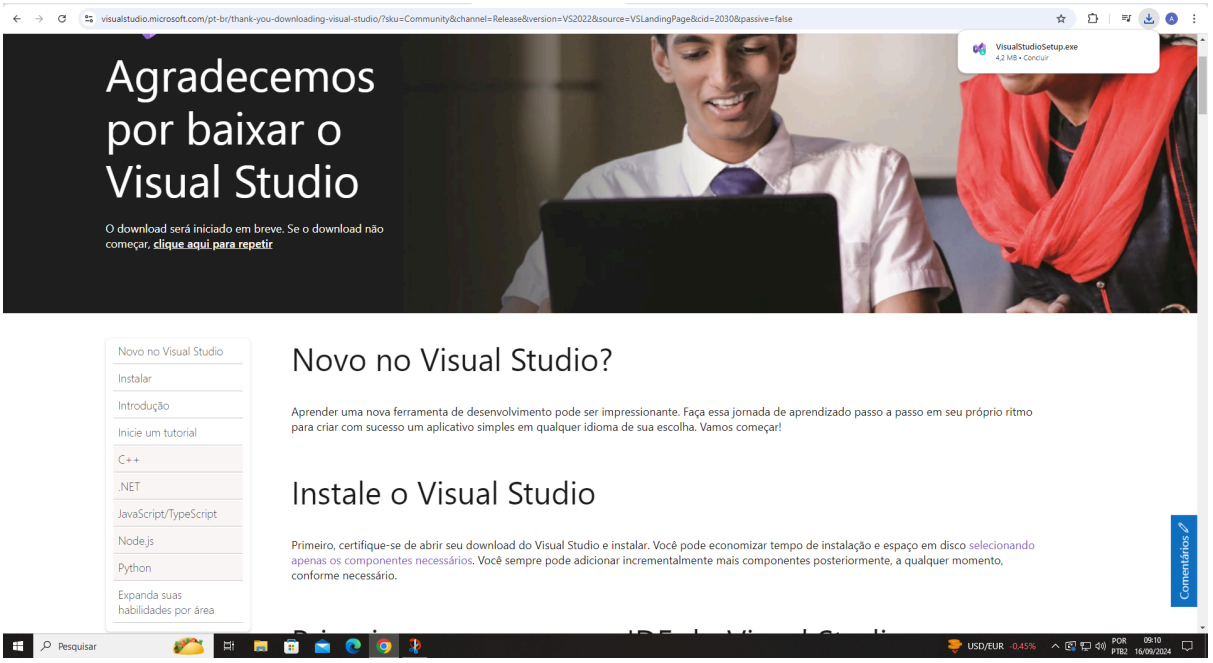
O Visual Studio Community 2022 é um editor de código avançado e otimizado para compilação de aplicativos Web e de nuvem modernos. Fornece suporte completo para desenvolvimento de uma web api Asp.Net por meio de bibliotecas e debugger.

Etapas de instalação

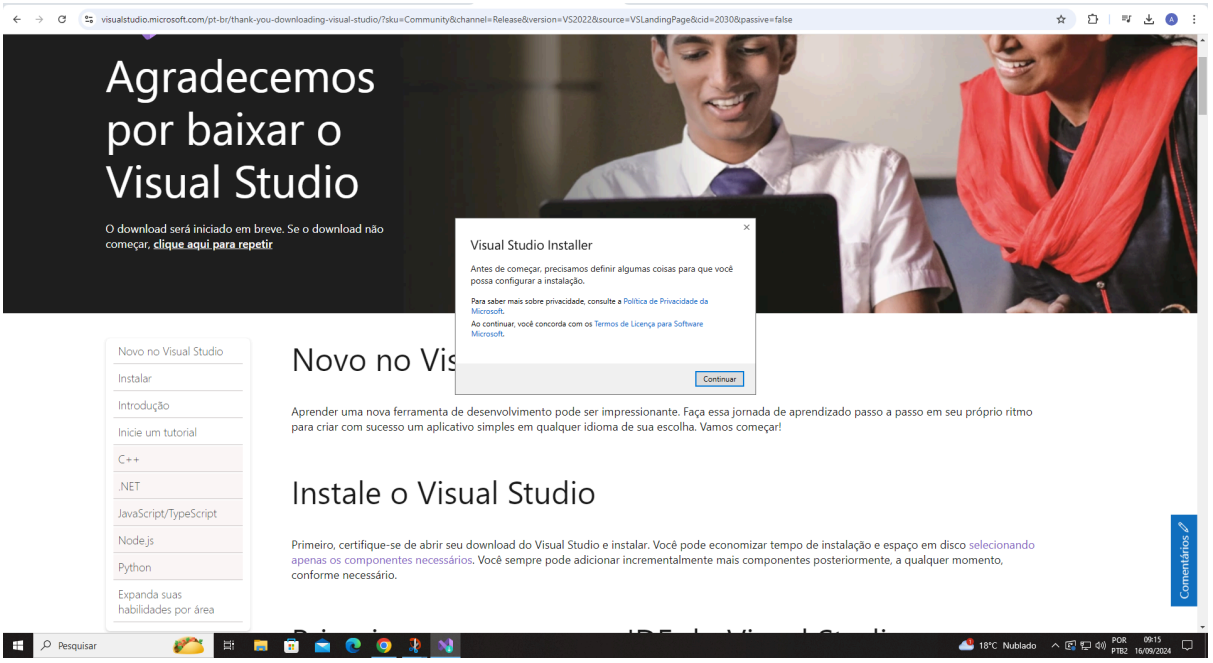
Para baixar o Visual Studio 2022 em sua máquina, primeiro acesse o site da Microsoft através do link: https://visualstudio.microsoft.com/pt-br/downloads/ e, depois, faça o download da versão do Sistema Operacional (SO) correspondente.



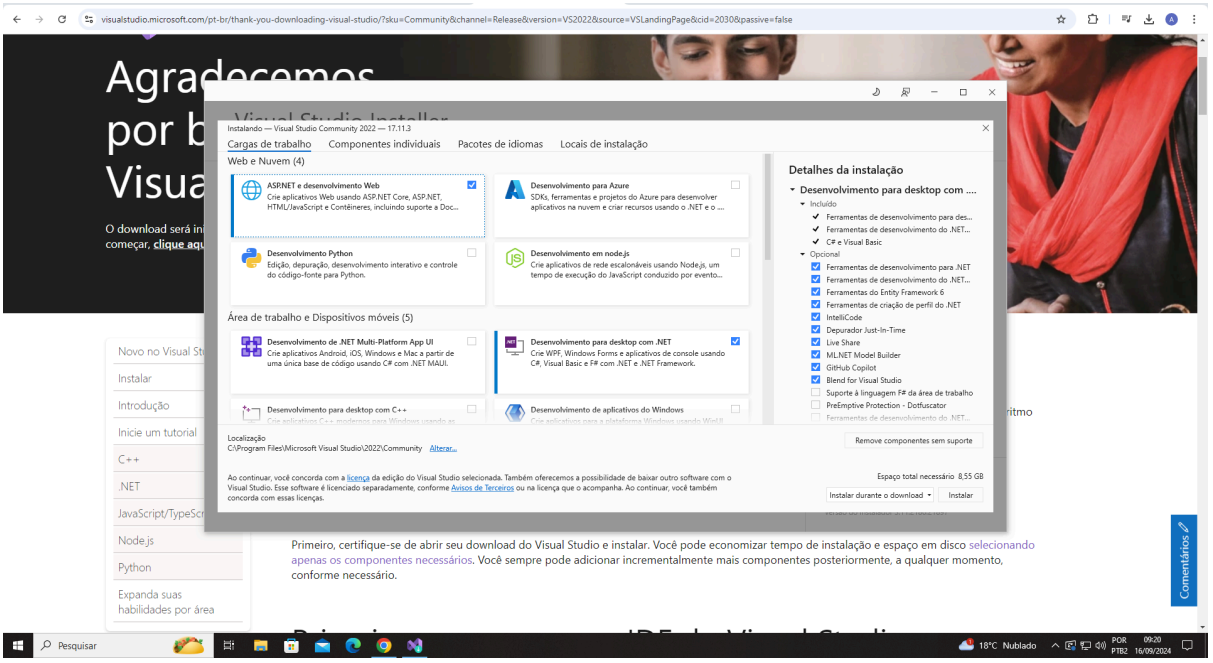
Após seguir as instruções iniciais, dê um duplo clique no download para prosseguir com a instalação da IDE.



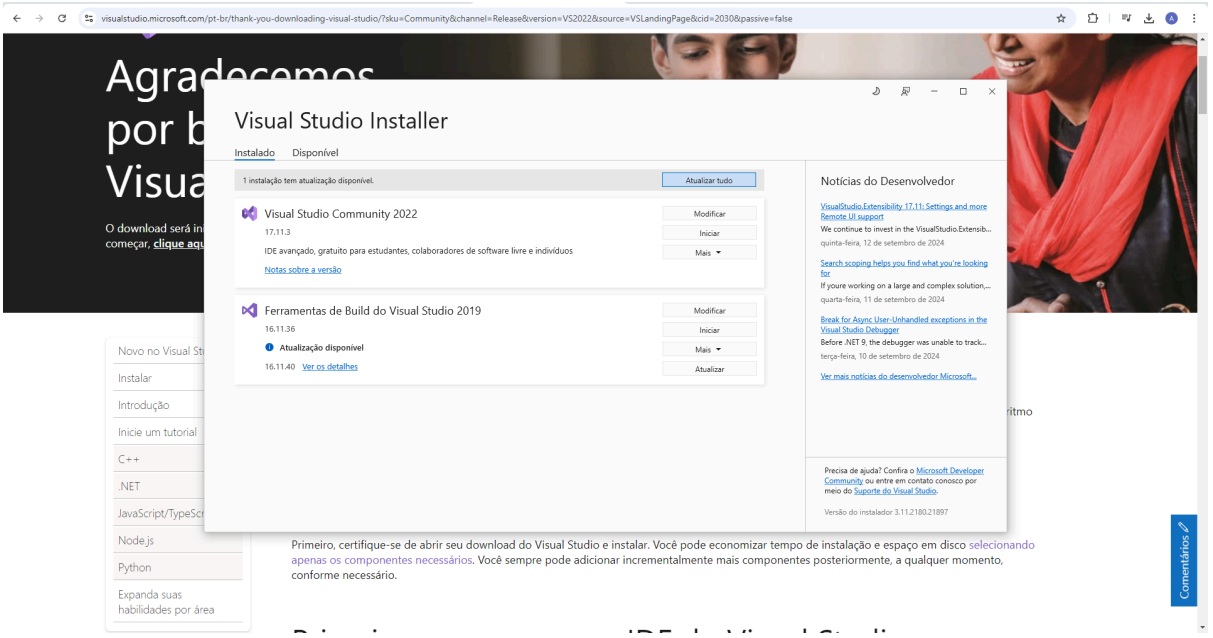
Antes de iniciar a instalação do Visual Studio 2022, o Instalador precisará das pré-configurações do programa, então clique em "Continuar" para realizá-las:



Após isso, será iniciada a tela com as pré-configurações do programa, selecione-as e clique em "Instalar"



Feito isso, o Instalador baixará as configurações selecionadas e instalará por fim o Visual Studio 2022.



Pronto! O Visual Studio Community 2022 está pronto para ser usado para o desenvolvimento de aplicações voltadas ao Back-End.

Front-End

Linguagem de programação

A linguagem de programação é um método padronizado, composto por um conjunto de regras sintáticas e semânticas, de implementação de um código-fonte — que pode ser compilado e transformado em um programa de computador. O Front-End do sistema Tech Saúde é inteiramente programado em JavaScript.

JavaScript

JavaScript (JS) é uma linguagem de programação interpretada estruturada, de script alto nível com tipagem dinâmica fraca e multipartidona (protótipos, orientado a objeto, imperativo e funcional). Atua juntamente com HTML e CSS para permitir que as páginas webs possam ser interativas e , portanto, é uma parte essencial dos aplicativos web.

Framework

Em desenvolvimento de software, o Framework é um conjunto de conceitos, ou melhor, uma abstração que une códigos comuns entre diferentes projetos provendo uma funcionalidade genérica. O Framework usado para ditar o fluxo de controle da aplicação Tech Saúde é o React.

React

O React permite a criação de interfaces de usuário a partir de partes individuais chamadas componentes. Trata-se de uma biblioteca Front-End JavaScript de código aberto com foco em criar interfaces de usuário em página web.

Vue.js

Vue.js é um framework JavaScript Front-end model-view-viewmodel de código aberto para construção de interfaces de usuário e aplicativos de página única. Baseia-se em HTML, CSS e JavaScript padrões com API intuitiva e documentação de alto nível.

Vite.js

Vite é um servidor de desenvolvimento local escrito e desenvolvido pelo mesmo criador do Vue.js. É usado

por padrão pelo Vue e para modelos de projeto React. Ele tem suporte para TypeScript e JSX. Ele usa Rollup e esbuild internamente para empacotamento.

Bibliotecas

Biblioteca na computação refere-se a uma coleção de subprogramas utilizados no desenvolvimento de software; estes contêm funções e dados auxiliares que podem ser usados em programas independentes, permitindo o compartilhamento e a alteração de código-fonte e dados modularmente. As bibliotecas usadas na aplicação foram o Bootstrap e a Axios.

Bootstrap

O Bootstrap é um Framework CSS de código aberto responsável pela estrutura Front-end. Na aplicação Tech Saúde, seu papel principal refere-se a aplicação de cor, tamanho, fonte e layout da biblioteca dele ao projeto.

Axios

O Axios permite a integração do projeto React para qualquer serviço de API disponível, e ganha destaque por se tratar de uma biblioteca leve e uma alternativa para quem usa o fetch API.

Código de Validação

A criação de pastas, reconhecida como componentes, ajuda na navegação e entendimento do código na hora de preparar as rotas, além da criação da página de navegação que facilita na hora da execução das validações.

Páginas de navegação

A criação da página main.jsx é importante para o entendimento de navegação das páginas ao clicar em algum botão que executa uma interação de troca de página ou surgimento de pop-ups.

Obs.: Você pode conferir os códigos HTML e a apresentação do Front-End no Anexo I.

Administrador

Este código React implementa um componente CRUD (Create, Read, Update, Delete) básico para gerenciar uma lista de médicos. Ele permite adicionar novos médicos, visualizar a lista existente, editar informações de médicos existentes e deletar médicos.

Importações:

- **React, useState**: Essenciais para criar componentes funcionais e gerenciar o estado (dados que podem mudar ao longo do tempo) em React.
- **FontAwesomeIcon:** Utilizado para adicionar ícones da biblioteca Font Awesome ao componente.
- **faEdit, faTrash, faTimes:** Ícones específicos da Font Awesome para as ações de editar, excluir e fechar.
- "./Crud.css": Importa um arquivo CSS para estilizar o componente.

Estado do Componente:

medicos: Um array que armazena os dados dos médicos, incluindo ID, nome, CRM e especialidade.
formData: Um objeto que armazena os dados do médico que está sendo editado ou adicionado.
isModalOpen: Um booleano que controla a abertura e fechamento do modal (janela modal).
isEdit: Um booleano que indica se o modal está em modo de edição ou criação.

Funções:

openModal: Abre o modal e preenche o formulário com os dados do médico a ser editado (se houver).
closeModal: Fecha o modal.
handleInputChange: Atualiza o estado do formulário quando o usuário digita algo nos campos de entrada.
handleSubmit: Salva as alterações do formulário, adicionando um novo médico ou atualizando um existente.
handleDelete: Remove um médico da lista.

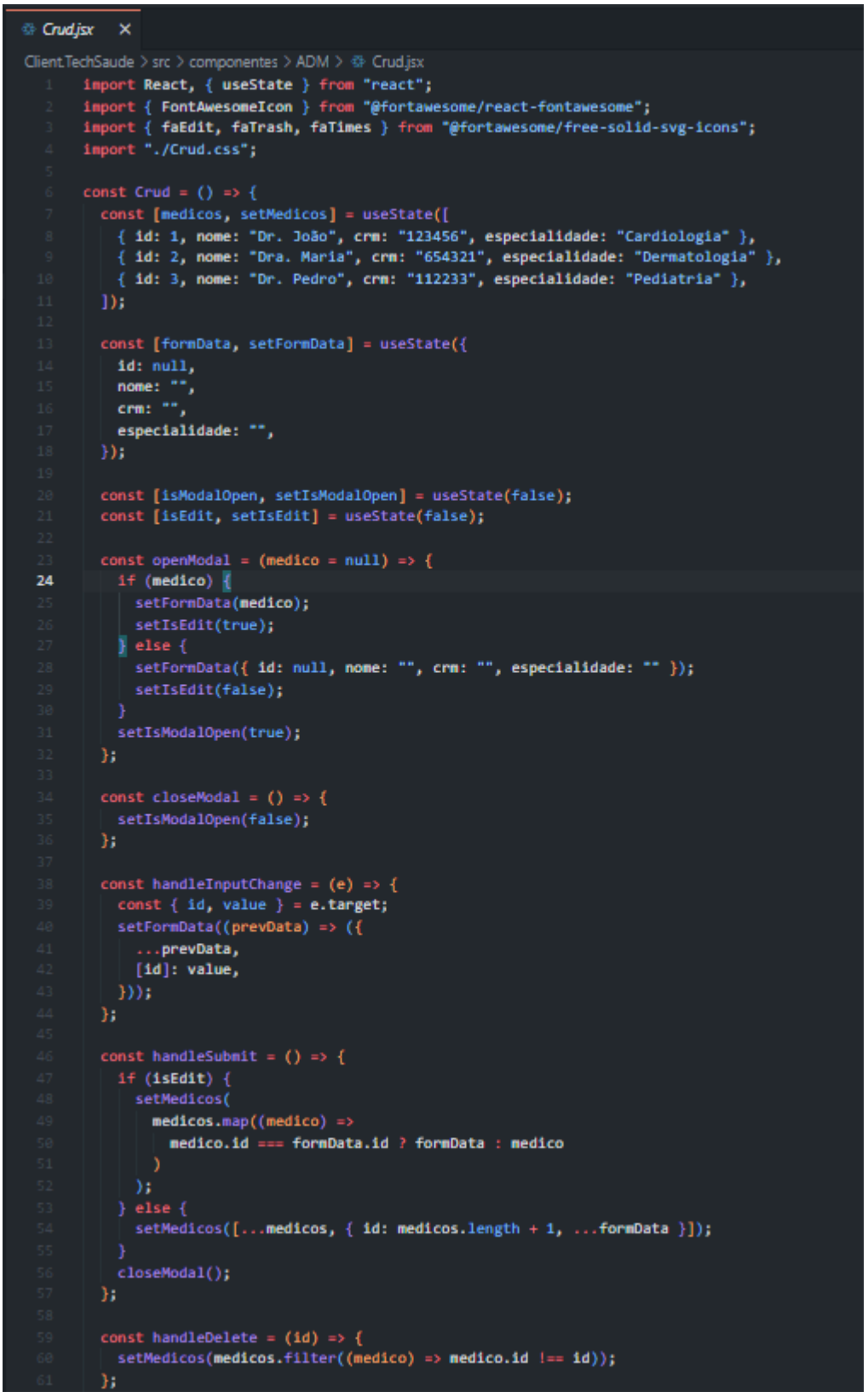
Fluxo do Código:

1. O componente renderiza inicialmente a lista de médicos.
2. Ao clicar no botão de editar ou no botão de adicionar, o modal é aberto e o formulário é preenchido com os dados do médico a ser editado ou deixado em branco para um novo médico.

3. O usuário pode editar os campos do formulário e clicar em salvar para confirmar as alterações.
4. Ao clicar no botão de excluir, o médico é removido da lista.

Pontos-chave:

useState: É usado extensivamente para gerenciar o estado do componente, como a lista de médicos, os dados do formulário e o estado do modal.
Modal: O modal é usado para exibir o formulário de edição/criação de médicos de forma não intrusiva.
CRUD: O componente implementa as operações básicas de um CRUD: criar, ler, atualizar e deletar.


Crud.jsx
ClientTechSaude > src > componentes > ADM > Crud.jsx
import React, { useState } from "react";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faEdit, faTrash, faTimes } from "@fortawesome/free-solid-svg-icons";
import "./Crud.css";
const Crud = () => {
const [medicos, setMedicos] = useState([
{ id: 1, nome: "Dr. João", crm: "123456", especialidade: "Cardiologia" },
{ id: 2, nome: "Dra. Maria", crm: "654321", especialidade: "Dermatologia" },
{ id: 3, nome: "Dr. Pedro", crm: "112233", especialidade: "Pediatria" },
]);
const [formData, setFormData] = useState({
id: null,
nome: "",
crm: "",
especialidade: "",
});
const [isModalOpen, setIsModalOpen] = useState(false);
const [isEdit, setIsEdit] = useState(false);
const openModal = (medico = null) => {
if (medico) {
setFormData(medico);
setIsEdit(true);
} else {
setFormData({ id: null, nome: "", crm: "", especialidade: "" });
setIsEdit(false);
}
setIsModalOpen(true);
};
const closeModal = () => {
setIsModalOpen(false);
};
const handleInputChange = (e) => {
const { id, value } = e.target;
setFormData((prevData) => ({
...prevData,
[id]: value,
}));
};
const handleSubmit = () => {
if (isEdit) {
setMedicos(
medicos.map((medico) =>
medico.id === formData.id ? formData : medico
)
);
} else {
setMedicos([...medicos, { id: medicos.length + 1, ...formData }]);
}
closeModal();
};
const handleDelete = (id) => {
setMedicos(medicos.filter((medico) => medico.id !== id));
};

Paciente

Vacina e Alergias

Ele importa diversos componentes e hooks do React e de uma biblioteca de componentes CSS (provavelmente react-bootstrap) para construir a interface do usuário relacionada a vacinas e alergias.

Importações

Linhas 1-4: Importa os componentes necessários do React e da biblioteca react-bootstrap. Isso inclui Form, Container, Dropdown, Modal, Button, useState, Sidebar e Header.

Linha 5: Importa um arquivo de estilos CSS personalizado para este componente.

Definição do Componente

Linha 7: Define um componente funcional chamado Alergias. Componentes funcionais são uma forma concisa de criar componentes em React.

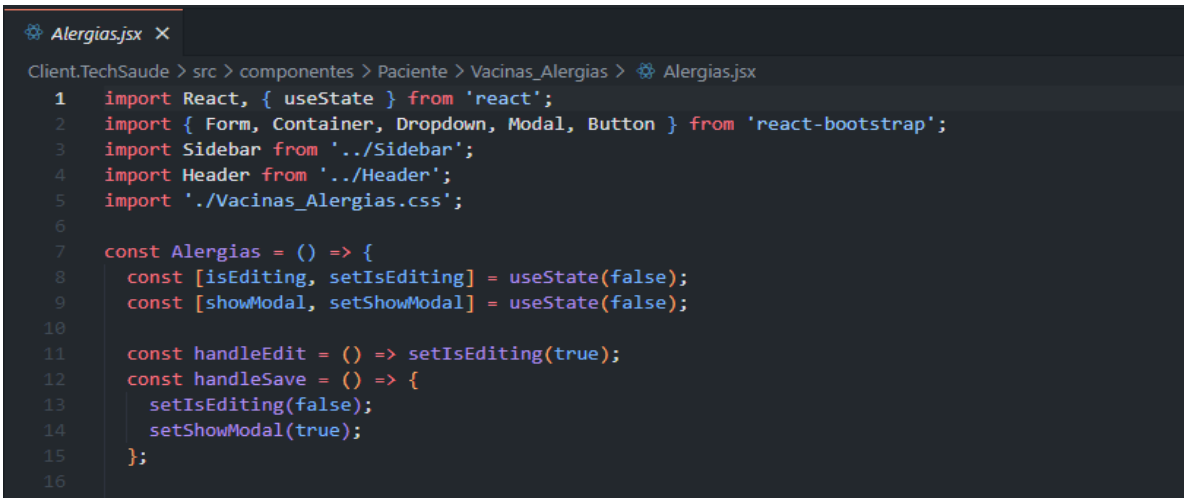
Linhas 8-10: Utiliza o hook useState para criar três estados:

isEditing: Um booleano que provavelmente controla se o componente está em modo de edição ou visualização.
showModal: Outro booleano que controla se um modal (uma caixa de diálogo) está sendo exibido.

Funções de Manipulação de Estado

Linhas 11-15: Define três funções para manipular os estados:

handleEdit: Define isEditing como true para entrar no modo de edição.
handleSave: Define isEditing como false e showModal como true para salvar as alterações e exibir um modal de confirmação.



Perfil

Definido o componente React chamado Perfil utilizado para permitir que um usuário edite suas informações pessoais em uma aplicação.

Importações:

React e Hooks: Importa os elementos básicos do React, como useState para gerenciar estados.
Componentes Bootstrap: Importa componentes do Bootstrap para criar a interface, como botões, formulários e modais.
Componentes Customizados: Importa Sidebar e Header, que provavelmente são componentes customizados da aplicação para a estrutura da página.
Perfil.css: Importa um arquivo CSS para estilizar o componente.

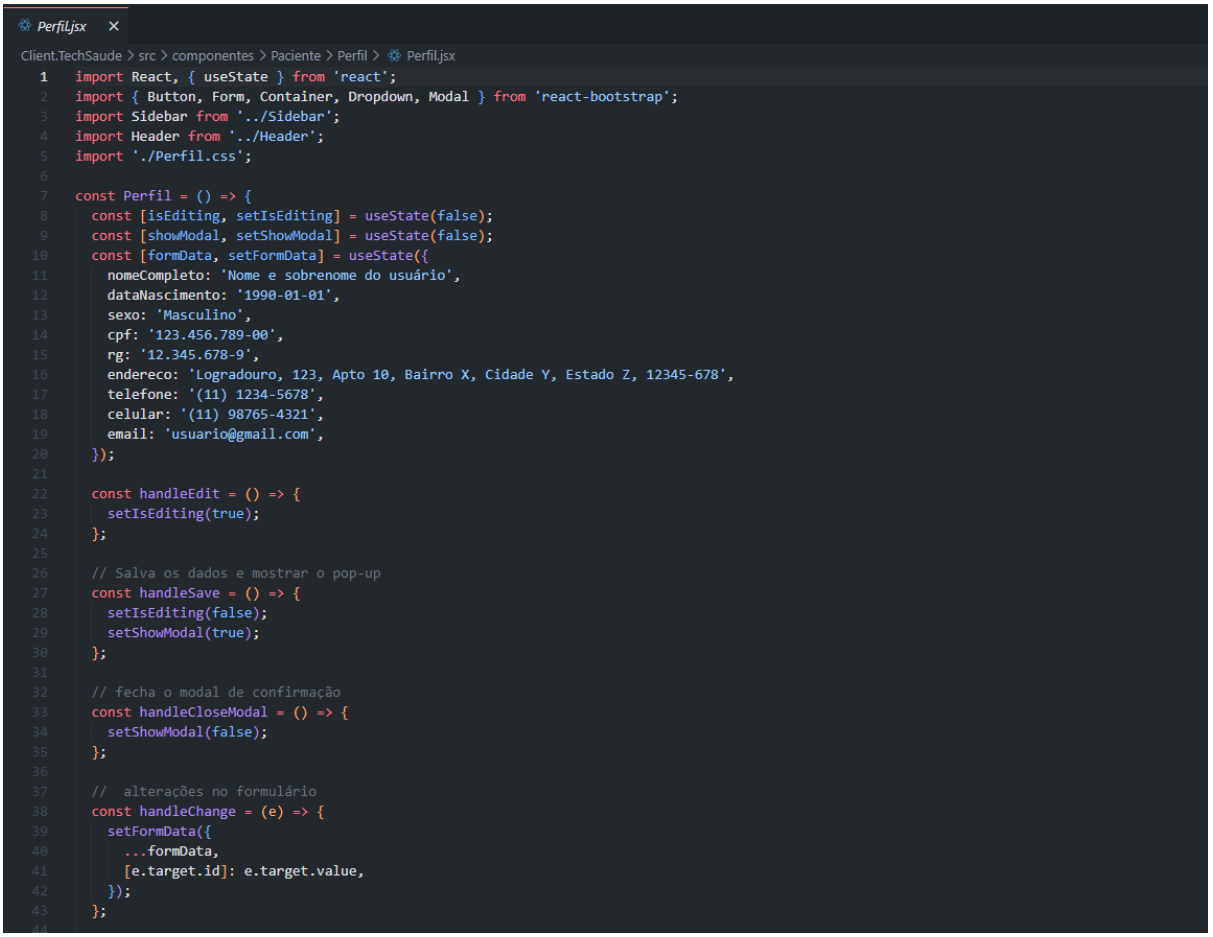
Estados:

isEditing: Indica se o componente está em modo de edição ou visualização.
showModal: Controla a visibilidade de um modal, possivelmente para confirmar a edição.
formData: Armazena os dados do formulário, como nome, data de nascimento, etc.

Funções:

handleEdit: Muda o estado para modo de edição, permitindo que o usuário altere os dados.
handleSave: Salva as alterações, enviando os dados para um servidor e mostrando um modal de confirmação.

handleCloseModal: Fecha o modal de confirmação.
handleChange: Atualiza o estado formData quando o usuário digita algo no formulário.



Histórico

No componente histórico encontram-se as páginas de receituário, consultas e de documentos.

Código da página de receituário:

Receituario.jsx

Client.TechSaude > src > componentes > Paciente > Historico > Receituario.jsx

```jsx
import React from 'react';
import { useNavigate } from 'react-router-dom';
import Sidebar from '../Sidebar';
import Header from '../Header';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faUpload } from '@fortawesome/free-solid-svg-icons';
import './Historico.css';

const Receituario = () => {
  const navigate = useNavigate();

  return (
    <div className="historico-container">
      <Sidebar />
      <div className="historico-main-content">
        <Header />
        <div className="historico-content">
          <div className="historico-buttons">
            <button onClick={() => navigate('/exames')} className="nav-button">Exames</button>
            <button onClick={() => navigate('/receituario')} className="nav-button">Receituários</button>
            <button onClick={() => navigate('/documentos')} className="nav-button">Documentos</button>
          </div>
          <ul className="historico-file-list">
            <li>Receituario1.pdf</li>
            <li>Receituario2.pdf</li>
          </ul>
          <div className="historico-upload-section">
            <label htmlFor="fileUpload" className="custom-file-upload">
              <FontAwesomeIcon icon={faUpload} /> Escolher Arquivo
            </label>
            <input type="file" className="form-control-file" id="fileUpload" />
            <div className="upload-controls">
              <button className="upload-btn">Enviar</button>
              <button className="cancel-btn">Cancelar</button>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Receituario;
```

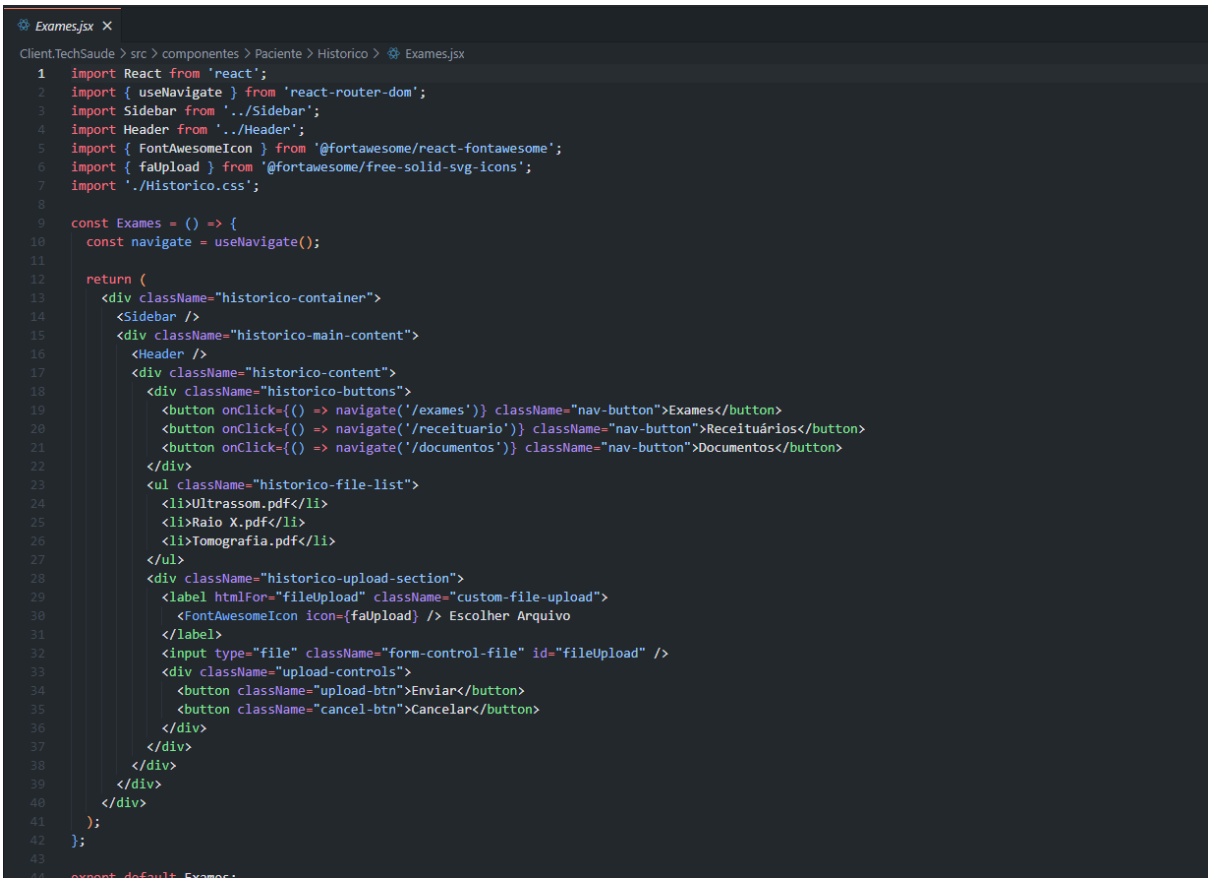
Código da página de documentos

Documentos.jsx

Client.TechSaude > src > componentes > Paciente > Historico > Documentos.jsx

```jsx
import React from 'react';
import { useNavigate } from 'react-router-dom';
import Sidebar from '../Sidebar';
import Header from '../Header';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faUpload } from '@fortawesome/free-solid-svg-icons';
import './Historico.css';

const Documentos = () => {
  const navigate = useNavigate();

  return (
    <div className="historico-container">
      <Sidebar />
      <div className="historico-main-content">
        <Header />
        <div className="historico-content">
          <div className="historico-buttons">
            <button onClick={() => navigate('/exames')} className="nav-button">Exames</button>
            <button onClick={() => navigate('/receituario')} className="nav-button">Receituários</button>
            <button onClick={() => navigate('/documentos')} className="nav-button">Documentos</button>
          </div>
          <ul className="historico-file-list">
            <li>Documento1.pdf</li>
            <li>Documento2.pdf</li>
          </ul>
          <div className="historico-upload-section">
            <label htmlFor="fileUpload" className="custom-file-upload">
              <FontAwesomeIcon icon={faUpload} /> Escolher Arquivo
            </label>
            <input type="file" className="form-control-file" id="fileUpload" />
            <div className="upload-controls">
              <button className="upload-btn">Enviar</button>
              <button className="cancel-btn">Cancelar</button>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Documentos;
```

Código da página de exames



```jsx
import React from 'react';
import { useNavigate } from 'react-router-dom';
import Sidebar from '../Sidebar';
import Header from '../Header';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faUpload } from '@fortawesome/free-solid-svg-icons';
import './Historico.css';

const Exames = () => {
  const navigate = useNavigate();

  return (
    <div className="historico-container">
      <Sidebar />
      <div className="historico-main-content">
        <Header />
        <div className="historico-content">
          <div className="historico-buttons">
            <button onClick={() => navigate('/exames')} className="nav-button">Exames</button>
            <button onClick={() => navigate('/receituario')} className="nav-button">Receituários</button>
            <button onClick={() => navigate('/documentos')} className="nav-button">Documentos</button>
          </div>
          <ul className="historico-file-list">
            <li>Ultrassom.pdf</li>
            <li>Raio X.pdf</li>
            <li>Tomografia.pdf</li>
          </ul>
          <div className="historico-upload-section">
            <label htmlFor="fileUpload" className="custom-file-upload">
              <FontAwesomeIcon icon={faUpload} /> Escolher Arquivo
            </label>
            <input type="file" className="form-control-file" id="fileUpload" />
            <div className="upload-controls">
              <button className="upload-btn">Enviar</button>
              <button className="cancel-btn">Cancelar</button>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};
```

Consultas

No componente consulta encontram-se as páginas de agendamento, agendadas e histórico.

Agendamento

Definindo um componente React chamado Agendamento que, com base no contexto e nas importações, é utilizado para permitir que um usuário agende consultas em uma aplicação médica.

Importações:

React e Hooks: Importa os elementos básicos do React, como useState para gerenciar estados.
Componentes Bootstrap: Importa componentes do Bootstrap para criar a interface, como Container, Form e Dropdown.
Componentes Customizados: Importa Sidebar e Header, que provavelmente são componentes customizados da aplicação para a estrutura da página.

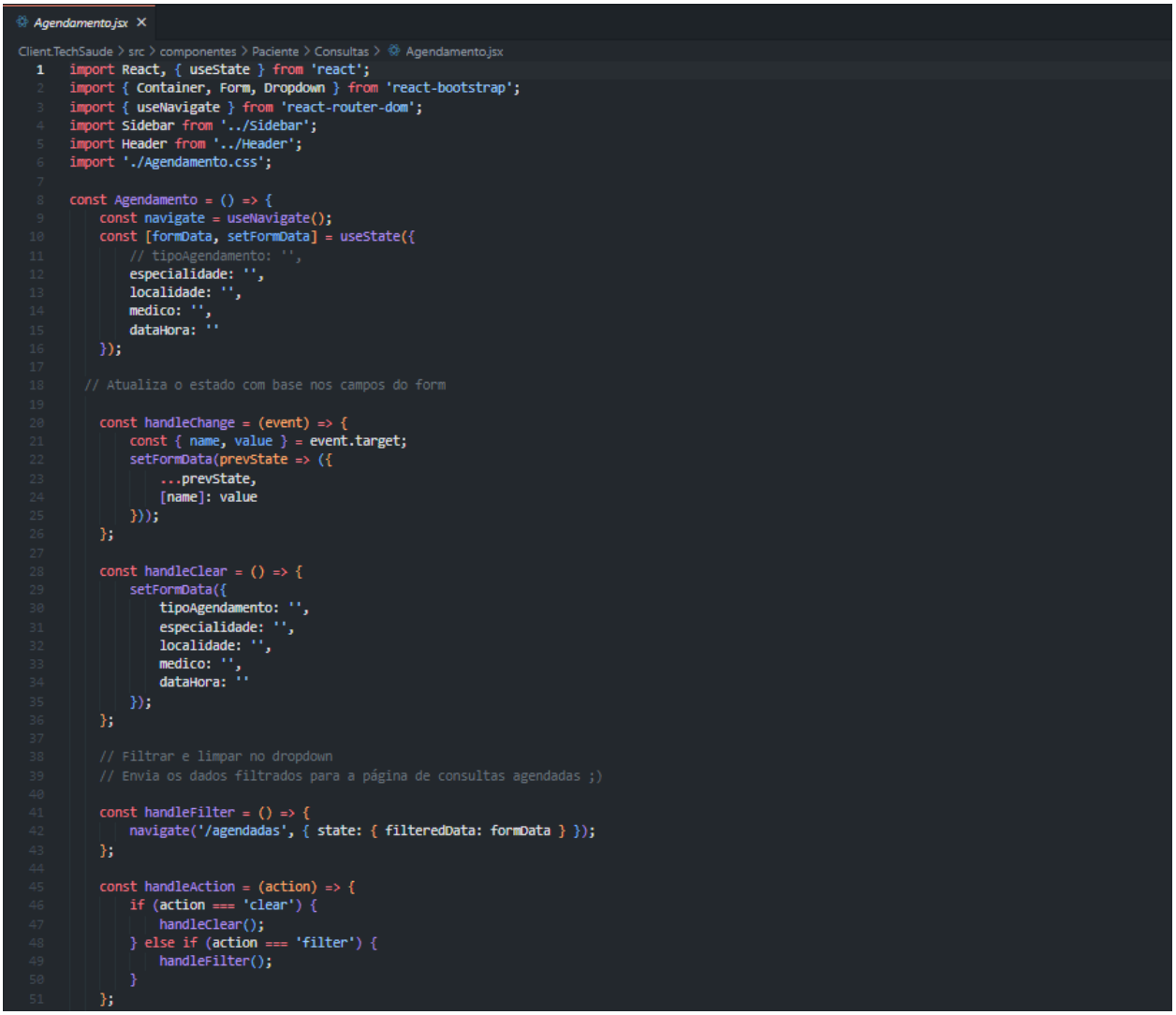
Agendamento.css: Importa um arquivo CSS para estilizar o componente.
useNavigate: Importa o hook useNavigate do react-router-dom para navegar entre as páginas.

Estados:

formData: Um objeto que armazena os dados do formulário de agendamento, como tipo de agendamento, especialidade, localidade, médico e data.

Funções:

handleChange: Atualiza o estado formData quando o usuário interage com os campos do formulário.
handleClear: Limpa todos os campos do formulário.
handleFilter: Filtra os dados do formulário, provavelmente para buscar consultas agendadas com base nos critérios selecionados.
handleAction: Chama as funções handleClear ou handleFilter com base na ação selecionada pelo usuário.
navigate: Navega para a página de consultas agendadas, passando os dados filtrados como parâmetro.



Agendadas

Definindo um componente React chamado Agendadas que, com base no contexto e nas importações, é utilizado para exibir uma lista de consultas agendadas em uma aplicação médica.

Importações:

React e Hooks: Importa os elementos básicos do React, como useState para gerenciar estados e useLocation para obter informações da URL.
Componentes Customizados: Importa Header, que provavelmente é um componente customizado da aplicação para o cabeçalho.
Agendadas.css: Importa um arquivo CSS para estilizar o componente.

Estados:

navigate: Função para navegar entre as páginas.
location: Objeto que contém informações sobre a URL atual, incluindo os parâmetros de pesquisa.
filteredData: Um objeto que armazena os dados de filtro, como tipo de agendamento, especialidade, localidade, médico e data.
agendadas: Um array que armazena os dados das consultas agendadas, simulando uma lista de dados reais.

Funções:

useEffect: Um hook que é executado após cada renderização e após as dependências mudarem. Neste caso, ele filtra a lista de agendamentos com base nos dados de filtro.
handleFilter: Filtra a lista de agendamentos com base nos dados de filtro.

Histórico

O código JavaScript apresentado parece ser parte de um componente React para uma aplicação web, provavelmente relacionada à área da saúde. Seu principal objetivo é manter um histórico de consultas médicas e adicionar novas consultas a esse histórico quando seu status se torna "Concluído".

Importações:

Componentes React: useState, useEffect para gerenciar o estado e efeitos colaterais, Container e Table do Bootstrap para estilização.

Rotas: useNavigate e useLocation para navegação entre páginas.

Componentes Customizados: Sidebar e Header (provavelmente componentes de navegação e cabeçalho da aplicação).

Estado Inicial:

Navegação: navigate e location são usados para controlar a navegação entre as páginas.

Consulta: consulta armazena os detalhes da consulta atual, obtidos da localização.

Histórico: histórico é um array que armazena as consultas concluídas, inicialmente com um objeto de exemplo.

useEffect Hook:

Verifica o status da consulta: Se o status da consulta atual for "Concluído", o código tenta adicionar a consulta ao histórico.

Verifica duplicidade: Antes de adicionar, verifica se a consulta já existe no histórico com base no ID.

Atualiza o histórico: Se a consulta não existir, ela é adicionada ao array histórico usando o spread operator para criar uma nova cópia do array.

Agendadas.jsx

Client.TechSaude > src > componentes > Paciente > Consultas > Agendadas.jsx

```jsx
import Header from '../Header';
import './Agendadas.css';

const Agendadas = () => {
    const navigate = useNavigate();
    const location = useLocation();
    const { filteredData } = location.state || { filteredData: {} };

    // Simulação de como ficaria os dados reais

    const [agendadas, setAgendadas] = useState([
        {
            id: 1,
            dataHora: '2024-06-25T14:30',
            medico: 'Dra. Mirella Dib Di Sessa',
            especialidade: 'Endocrinologia',
            localidade: 'Santos',
            status: 'Agendado',
        },
        {
            id: 2,
            dataHora: '2024-06-25T15:00',
            medico: 'Dra. Ana Silva',
            especialidade: 'Cardiologia',
            localidade: 'Praia Grande',
            status: 'Agendado',
        },
        {
            id: 3,
            dataHora: '2024-06-25T16:00',
            medico: 'Dra. João Pereira',
            especialidade: 'Neurologia',
            localidade: 'São Vicente',
            status: 'Agendado',
        },
    ]);

    // Filtra as consultas agendadas com base nos dados do form

    useEffect(() => {
        if (filteredData) {
            setAgendadas(prevState =>
                prevState.filter(agendamento =>
                    (!filteredData.tipoAgendamento || agendamento.tipoAgendamento === filteredData.tipoAgendamento) &&
                    (!filteredData.especialidade || agendamento.especialidade === filteredData.especialidade) &&
                    (!filteredData.localidade || agendamento.localidade === filteredData.localidade) &&
                    (!filteredData.medico || agendamento.medico === filteredData.medico) &&
                    (!filteredData.dataHora || agendamento.dataHora === filteredData.dataHora)
                )
            );
        }
    }, [filteredData]);

    // Confirmar ou cancelar uma consulta no dropdown
    // Poderiamos ter algum tipo de verificação automatica para o redirecionamento
```

```jsx
    const handleAction = (id, action) => {
        if (action === 'confirmar') {
            const consulta = agendadas.find(c => c.id === id);
            if (consulta) {
                consulta.status = 'Concluído';
                // Envia a consulta para o histórico
                navigate('/historico', { state: { consulta } });
            }
        } else if (action === 'cancelar') {
            setAgendadas(prevState => prevState.filter(c => c.id !== id));
        }
    };
```

Cadastro

Validação Data de nascimento

O código fornece uma validação robusta para a data de nascimento, garantindo que os dados inseridos pelo usuário sejam consistentes e possam ser processados corretamente pelo sistema. Ele tem como objetivo validar a data de nascimento inserida pelo usuário em um formulário, garantindo que esteja no formato correto (dd/mm/aaaa) e que seja uma data válida.

Importações:

React e dependências: Importa os componentes e hooks necessários para criar um componente React.
Axios: Provavelmente utilizado para realizar requisições HTTP para uma API, possivelmente para salvar os dados do paciente.
Bootstrap e Font Awesome: Importa estilos CSS para dar um visual mais profissional ao componente.
InputMask: Um componente externo utilizado para formatar a entrada do usuário no formato de data.

Variáveis e Constantes:

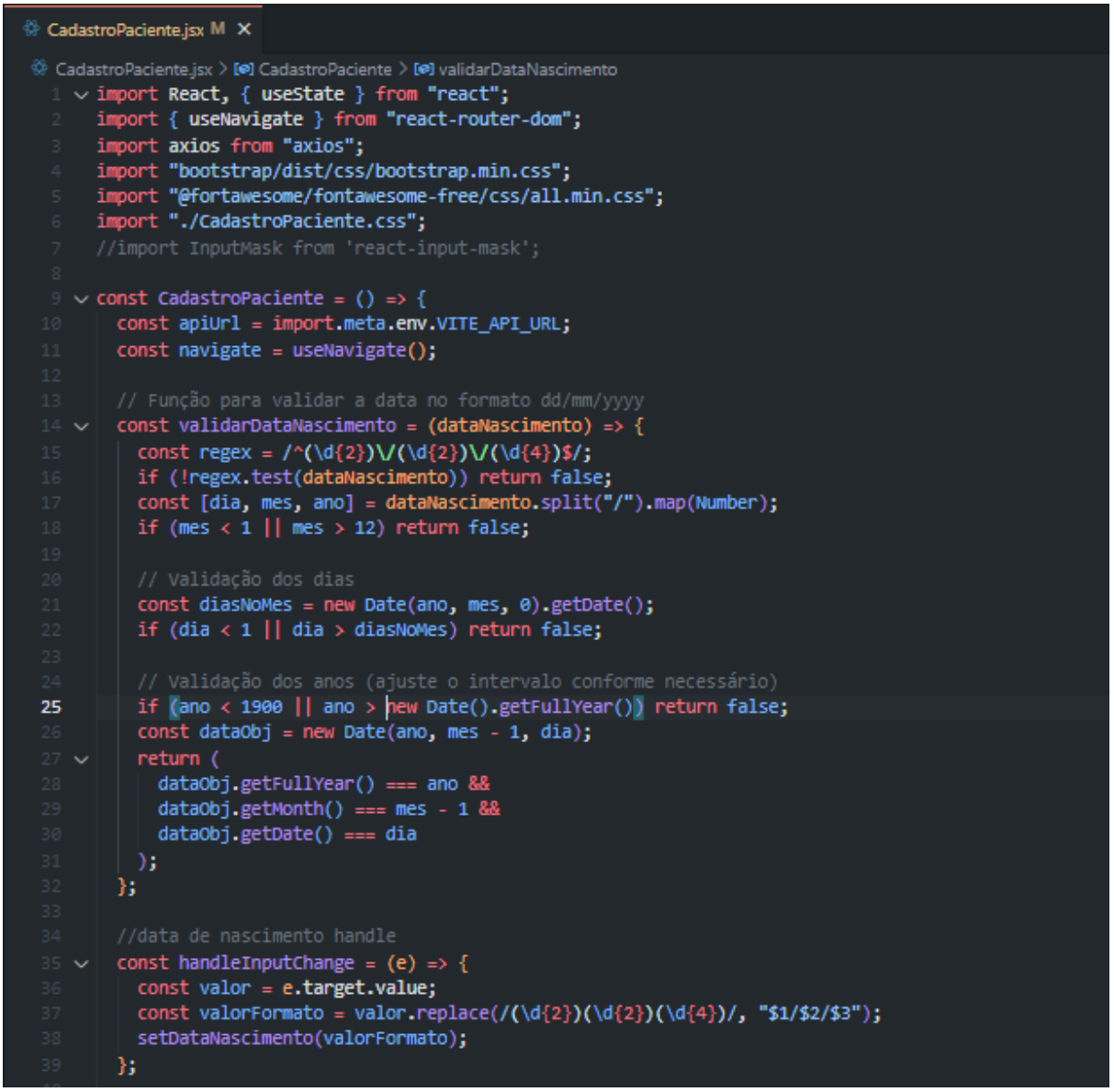
apiUrl: Armazena a URL da API que será utilizada para enviar os dados do paciente.
navigate: Uma função para navegar entre páginas, provavelmente para redirecionar o usuário após o cadastro.

Função validarDataNascimento:

Expressão regular: Utiliza uma expressão regular para verificar se a data está no formato dd/mm/aaaa.
Separação dos componentes: Separa o dia, mês e ano da data para realizar validações mais específicas.
Validação do mês: Verifica se o mês está entre 1 e 12.
Validação dos dias: Calcula o número de dias do mês e verifica se o dia inserido está no intervalo válido.
Validação do ano: Define um intervalo de anos válidos (ajustável) e verifica se o ano inserido está dentro desse intervalo.
Criação de um objeto Date: Cria um objeto Date para realizar verificações adicionais e garantir que a data seja válida.

handleInputChange:

Formatação da entrada: Formata a data inserida pelo usuário no formato dd/mm/aaaa à medida que o usuário digita.
Atualização do estado: Atualiza o estado do componente com a data formatada.



Validação Formatações de campo

Este código estabelece a base para criar um formulário de cadastro interativo. As variáveis de estado permitem que o componente React rastreie as mudanças nos dados do formulário e atualize a interface do usuário de acordo.

Definido e inicializa variáveis de estado que serão utilizadas para gerenciar os dados do formulário de cadastro de um paciente em uma aplicação React. Essas variáveis armazenam informações como o nome completo, email, senha, data de nascimento, telefone e o estado de outros elementos da interface, como botões e mensagens de erro.

Importando o React: O código assume que você já importou o React e outros componentes necessários para criar um componente funcional.
Definindo o Componente: O componente CadastroPaciente é definido e dentro dele, as variáveis de estado são declaradas.
useState Hook: O hook useState do React é utilizado para criar e gerenciar cada uma das variáveis de estado. Cada chamada a useState retorna um array com dois valores:
O valor atual da variável de estado.
Uma função para atualizar o valor da variável.

Variáveis de Estado:

1. formData: Um objeto que armazena todos os dados do formulário, como nome, email, senha, etc.
2. errors: Um objeto que armazena possíveis mensagens de erro para cada campo do formulário.
3. loading, isSubmitting: Variáveis booleanas utilizadas para controlar o estado de carregamento e envio do formulário.
4. senha, confirmaSenha, showSenhaMessage: Variáveis relacionadas à senha do usuário, incluindo uma mensagem para indicar se as senhas coincidem.
5. dataNascimento, telefone: Variáveis para armazenar a data de nascimento e o número de telefone do usuário.
6. checkbox1, checkbox2, buttonEnabled: Variáveis para controlar o estado de checkboxes e habilitar/desabilitar botões no formulário.

```
const CadastroPaciente = () => {
  const apiUrl = import.meta.env.VITE_API_URL;
  const navigate = useNavigate();

  const [formData, setFormData] = useState({
    nomeCompleto: "",
    email: "",
    senha: "",
    confirmarSenha: "",
    dataNascimento: "",
    cns: "",
    telefone: "",
    termo: false,
    compartilhamento: false,
  });

  const [erros, setErros] = useState({});
  const [loading, setLoading] = useState(false);
  const [isSubmitting, setIsSubmitting] = useState(false);


  const [showSenhaMessage, setShowSenhaMessage] = useState(false);
  const [buttonEnabled, setButtonEnabled] = useState(false);
```

Funções de validação

Definindo um conjunto de funções de validação para verificar a integridade dos dados inseridos em um formulário, provavelmente um formulário de cadastro. Essas funções garantem que os dados estejam no formato correto e atendam a determinados critérios.

Explicação Detalhada das Funções:

validarNome(nomeCompleto):

Utiliza uma expressão regular para verificar se o nome completo contém apenas letras, espaços e alguns caracteres especiais como hífens.

Retorna true se o nome for válido e false caso contrário.

validarEmail(email):

Emprega uma expressão regular mais complexa para validar o formato do endereço de email.

Verifica se o email contém o caractere '@', um ponto ('.') após o '@' e se não há espaços em branco.

Retorna true se o email for válido e false caso contrário.

validarSenhas():

Compara a senha com a confirmação da senha.
Retorna true se as senhas forem iguais e false caso contrário.

validarSenhasEConfirmarSenha():

Verifica se as senhas são iguais e se a senha atende a critérios de força (não implementados no código fornecido).
Se as senhas forem diferentes, adiciona uma mensagem de erro ao objeto erros.
Se a senha não atender aos critérios de força, adiciona outra mensagem de erro ao objeto erros.

validarCns(cns):

Utiliza uma expressão regular para verificar se o número do Cartão Nacional de Saúde (CNS) possui 15 dígitos.
Retorna true se o CNS for válido e false caso contrário.

validarTelefone(telefone):

Emprega uma expressão regular para verificar se o número de telefone está no formato (DD) XXXXX-XXXX.
Se o número não estiver no formato correto, retorna o número original.
Caso contrário, formata o número e o retorna.

Chamadas das Funções: Ao enviar o formulário, essas funções são chamadas para validar cada campo.
Verificação das Condições: Cada função aplica as regras de validação específicas para o campo correspondente.
Retorno de Resultados: As funções retornam true se a validação for bem-sucedida e false ou uma mensagem de erro caso contrário.

Gerenciamento de Erros: As mensagens de erro retornadas pelas funções são geralmente armazenadas em um objeto erros e exibidas ao usuário para que ele possa corrigir os dados inválidos.

```
69    // Funções de validação
70    const validarNome = (nomeCompleto) =>
71      /^[A-Za-zÀ-Ö0-ÿ\s]+$/.test(nomeCompleto);
72    const validarEmail = (email) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
73
74    const validarSenhas = () => {
75      return senha === confirmaSenha;
76    };
77    const validarConfirmaSenha = (senha, confirmarSenha) => {
78      if (senha !== confirmarSenha) {
79        return false;
80      }
81      return true;
82    };
83
84    const validarCns = (cns) => /^\d{15}$/.test(cns);
85
86    const validarTelefone = (telefone) => {
87      const regex = /^\(?([0-9]{2})\)?[-. ]?([0-9]{5})[-. ]?([0-9]{4})$/;
88      if (!regex.test(telefone)) {
89        return telefone; // retorna o telefone original se não estiver no formato correto
90      }
91      const formattedTelefone = telefone.replace(regex, "($1) $2-$3");
92      return formattedTelefone;
93    };
94
95    const validarSenhaEConfirmarSenha = () => {
96      if (!validarSenhas(senha)) {
97        erros.senha =
98          "A senha deve ser forte: símbolo, número e letra maiúscula.";
99      }
100     if (senha !== confirmaSenha) {
101       erros.confirmarSenha = "As senhas não conferem.";
102     }
103   };
```

```
9   const CadastroPaciente = () => {
105   const handleSubmit = async (e) => {
106     e.preventDefault();
107
108     if (isSubmitting) return;
109
110     // Nova lista de erros
111     const errosValidacao = {};
112
113     // Validações
114     if (!validarNome(formData.nomeCompleto))
115       errosValidacao.nomeCompleto =
116         "O nome não deve conter símbolos ou números.";
117     if (!validarEmail(formData.email))
118       errosValidacao.email = "O e-mail fornecido é inválido.";
119     if (!validarSenhas(formData.senha))
120       errosValidacao.senha =
121         "A senha deve ser forte: símbolo, número e letra maiúscula.";
122     if (!validarConfirmaSenha(senha, confirmaSenha)) {
123       errosValidacao.confirmarSenha = "As senhas não conferem.";
124     }
125     if (!validarDataNascimento(formData.dataNascimento))
126       errosValidacao.dataNascimento =
127         "A data deve estar no formato dd/mm/yyyy e ser uma data válida.";
128     if (!validarCns(formData.cns))
129       errosValidacao.cns = "O CNS deve ter 15 dígitos.";
130     if (!validarTelefone(formData.telefone))
131       errosValidacao.telefone = "O número de telefone precisa ter 11 números.";
132     if (!validarSenhaEConfirmarSenha()) {
133       return;
134     }
135
136     // Atualiza o estado de erros
137     setErros(errosValidacao);
138
139     if (Object.keys(erros).length > 0) {
140       setErros(erros);
141       return;
142     }
143
144     try {
145       const response = await axios.post(`${apiUrl}/api/Paciente/register`);
146       console.log(response);
147       if (response.data.sucesso) {
148         alert(response.data.message);
149         navigate("/perfil");
150       }
151     } catch (error) {
152       console.error("Erro ao enviar dados:", error);
153       alert("Ocorreu um erro ao enviar os dados. Por favor, tente novamente.");
154     }
155   };
```

```
157   const checkButtonEnable = () => {
158     if (!checkbox1 && !checkbox2) {
159       setButtonEnabled(false);
160       console.log("Checkbox Termos Confirmada");
161     } else {
162       setButtonEnabled(true);
163       console.log("Checkbox Compartilhar Confirmada");
164     }
165   };
166
167   const handleCheckboxChange = (checkbox, value) => {
168     if (checkbox === "checkbox1") {
169       setCheckbox1(value);
170     } else if (checkbox === "checkbox2") {
171       setCheckbox2(value);
172     }
173     checkButtonEnable();
174   };
```

Back-End

Linguagem de programação

C#

É uma linguagem de programação de alto nível de uso geral que oferece suporte a vários paradigmas. C# abrange disciplinas de programação estática, tipagem forte, escopo lexical, imperativo, declarativo, funcional, genérico, orientado a objetos (baseado em classe) e orientado a componentes. Paradigma C#.

Framework

Asp.Net

É uma estrutura de aplicativo da Web do lado do servidor projetada para desenvolvimento Web para produzir páginas Web dinâmicas. Foi desenvolvido pela Microsoft para permitir que programadores construam sites, aplicativos e serviços dinâmicos. O nome significa *Active Server Pages Network Enabled Technologies*.

Bibliotecas

Entity Framework (EF)

É uma estrutura de mapeamento objeto-relacional (ORM) de código aberto que permite aos desenvolvedores trabalhar com dados relacionais. Ele é usado para criar camadas de acesso a dados para .NET (C#) em vários bancos de dados.

JSON Web Tokens (JWT)

É um padrão da Internet proposto para a criação de dados com assinatura opcional e/ou criptografia opcional cuja carga útil contém JSON que afirma um certo número de declarações. Os tokens são assinados usando um segredo privado ou uma chave pública/privada.

Conceitos

- Repository Pattern
- Code First
- Api Restful

Código

Para o desenvolvimento da Web API ASP.NET para a aplicação TECHSAUDE, foi necessária a instalação de algumas bibliotecas como:

- **SQL Server** (Microsoft.EntityFrameworkCore.SqlServer) para integração com bancos de dados,
- **Entity Framework Core** (Microsoft.Entity Framework Core) para o mapeamento objeto-relacional
- **ASP.NET Core Identity** (Microsoft.AspNetCore.Identity) para o gerenciamento de autenticação e usuários.
- **Tools** (Microsoft.Entity Framework Core.Tools) para facilitar a manipulação de bancos de dados
- **Design** (Microsoft.Entity Framework Core.Design) são utilizadas para gerar migrações e executar comandos.

Além disso, para a autenticação via tokens JWT, foi instalado o pacote JwtBearer (Microsoft.AspNetCore.Authentication.JwtBearer) e para documentar a API e criar uma interface interativa, o Swagger (Swashbuckle.Asp Net Core) foi instalado, facilitando a visualização e teste das rotas da API. Essas bibliotecas podem ser facilmente instaladas por meio do NuGet ou da CLI do .NET, como mostra o **Anexo 1 - BackEnd**.

Conexão com o banco de dados

Após a instalação das bibliotecas necessárias, como o Entity Framework Core e o SQL Server, o próximo passo é criar um arquivo chamado AppDbContext, configurá-lo para a conexão com o banco de dados SQL Server e pelo mapeamento das tabelas a partir do Entity

Framework. O AppDbContext é uma classe que herda de DbContext e define as entidades que serão mapeadas para as tabelas no banco de dados. Essa classe permite que você interaja com o banco de dados de forma mais estruturada e organizada, utilizando a abstração do ORM (Object-Relational Mapping).

Diante disso, o appsettings.json é configurado com as informações de conexão com o banco de dados, definindo a string de conexão, que especifica o servidor, o nome do banco de dados e as credenciais necessárias para acessar o banco. O arquivo appsettings.json fornece um local centralizado para gerenciar a configuração da conexão com o banco de dados, facilitando a manutenção e a modificação desses parâmetros conforme necessário.

Desse modo, com AppDbContext configurado para mapear as entidades e a string de conexão definida no appsettings.json, a aplicação estará pronta para interagir com o banco de dados SQL Server usando o Entity Framework Core. Conforme ilustrado no **Anexo 2 - BackEnd.**

Configuração do Program.cs

A seguir, será necessário a configuração do arquivo **Program.cs** para estabelecer a base da aplicação ASP.NET Core, incluindo a comunicação com o banco de dados, a autenticação e autorização de usuários, a segurança e a documentação da API. Essas configurações são cruciais para garantir que a aplicação seja segura, eficiente e bem estruturada. O Anexo 3 - BackEnd apresenta a configuração utilizada na aplicação.

Modelos

No contexto de uma aplicação ASP.NET Core com Entity Framework Core, os modelos são classes que representam as entidades ou tabelas do banco de dados. Cada modelo é mapeado para uma tabela no banco de dados, e suas propriedades correspondem às colunas dessa tabela. O Entity Framework Core utiliza essas classes para realizar operações de **CRUD** (criação, leitura, atualização e exclusão) de maneira simplificada, permitindo a interação com o banco de dados orientado a objetos.

Conforme o Anexo 4 (Back-End), que mostra os modelos utilizados para a criação do TecSaúde.

Controllers

O Controller em uma Web API no ASP.NET Core é uma classe que gerencia as requisições HTTP recebidas, processa essas requisições e retorna respostas adequadas. Ele serve como ponto de entrada para as operações da API, ou seja, é a camada que recebe as solicitações do cliente e as direciona para os métodos que manipulam os dados, como buscar, criar, atualizar ou excluir informações. Os controllers em uma Web API são responsáveis por implementar a lógica de processamento de requisições, sem necessidade de interação com uma interface gráfica, como em um aplicativo MVC tradicional.

No Anexo 5 são apresentados os principais controller criados para o funcionamento da web API:

Repository Pattern

O **Repository Pattern** (Padrão de Repositório) é um padrão de design que visa separar a lógica de acesso a dados da lógica de negócio de uma aplicação. Ele abstrai a camada de persistência (como bancos de dados ou outros mecanismos de armazenamento) e permite que você trabalhe com objetos de domínio diretamente, sem se preocupar com os detalhes de implementação da base de dados.

Principais Características do Repository Pattern:

1. **Abstração**: O repositório oferece uma interface que abstrai os detalhes de implementação de acesso a dados. Isso significa que a lógica de negócios pode interagir com o repositório sem saber se os dados estão sendo recuperados de um banco de dados relacional, um arquivo ou uma API externa.
2. **Encapsulamento de Acesso a Dados**: O repositório encapsula a lógica de acesso aos dados. Ele esconde a complexidade de consultas e operações de **CRUD** (Create, Read, Update, Delete) e apresenta uma interface simples para a aplicação.
3. **Manutenção e Testabilidade**: Como a lógica de acesso a dados é centralizada no repositório, fica mais fácil manter e modificar o código. Além disso, permite realizar testes unitários na lógica de negócios sem depender de um banco de dados real, utilizando mocks ou stubs do repositório.

Vantagens:

1. **Desacoplamento**: Permite que a camada de lógica de negócios seja desacoplada da lógica de persistência.

2. **Facilidade de Manutenção**: Alterações no mecanismo de persistência (como mudar de banco de dados) podem ser feitas com impacto mínimo na aplicação.
3. **Testabilidade**: Facilita o teste da lógica de negócios com o uso de mocks para o repositório.

Repositório da Alergias

Conforme o Anexo 6 (BackEnd), o repositório de alergias é responsável por gerenciar as informações sobre as alergias dos pacientes. Ele inclui funções como AddAlergias, que permite adicionar novas alergias ao histórico médico de um paciente; GetAlergias, que recupera as alergias registradas para um paciente específico; e UpdateAlergias, que possibilita a atualização das alergias já cadastradas. Essas funções são fundamentais para manter um registro completo e atualizado das alergias dos pacientes, proporcionando uma gestão mais eficaz e segura do histórico médico.

Funções:

AddAlergia: A função AddAlergia adiciona uma nova alergia ao histórico médico de um paciente. Ela busca o histórico médico associado ao idPaciente no banco de dados e valida se o campo Descricao da alergia foi fornecido. Caso o histórico não seja encontrado ou a descrição esteja ausente, retorna mensagens de erro apropriadas. Se válido, cria uma nova instância de Alergia com os dados do modelo AlergiasDTO, adiciona-a ao histórico médico, e salva as alterações no banco de dados. Em seguida, retorna um objeto AlergiaSaidaDTO com as informações da alergia criada e uma mensagem de sucesso. Em caso de falhas, registra o erro e retorna uma mensagem genérica indicando o problema. Conforme o Anexo 6.1 (BackEnd)

GetAlergia : A função GetAlergias busca o histórico médico de um paciente pelo seu ID no banco de dados, utilizando o Entity Framework para incluir a lista de alergias relacionadas. Se nenhum histórico for encontrado, retorna uma mensagem indicando a ausência de dados. Caso o histórico seja localizado, as alergias vinculadas a ele são convertidas em objetos AlergiaSaidaDTO, contendo informações como ID, descrição e medicamento. Esses objetos são então retornados em uma lista acompanhada de uma mensagem de sucesso. Em caso de erro, o

problema é registrado no log e uma resposta genérica de falha é enviada ao cliente. Conforme o Anexo 6.2 (BackEnd)

UpdateAlergia : A metodo UpdateAlergia realiza a atualização de uma alergia no banco de dados. Primeiro, utiliza o Entity Framework para buscar a alergia correspondente ao idAlergia fornecido. Se nenhuma alergia for encontrada, retorna uma resposta de falha informando que o registro não foi localizado. Caso encontre o registro, os campos Descricao e Medicamento da alergia são atualizados com os valores do objeto AlergiasDTO fornecido como entrada. Em seguida, as alterações são salvas no banco de dados utilizando o método SaveChangesAsync. Após isso, um objeto AlergiaSaidaDTO é criado para incluir os dados atualizados na resposta de sucesso. Se ocorrer algum erro durante o processo, a função captura a exceção, registra o erro no log e retorna uma mensagem de falha genérica ao cliente. Conforme o Anexo 6.3 (BackEnd)

Repositório da Autenticação

Abaixo a foto mostra o código de um controller de autenticação (AuthController) em uma aplicação Web API construída com ASP.NET Core. O código define um endpoint HTTP POST para o login de usuários, onde as credenciais (email e senha) são recebidas e validadas por meio de um repositório (AuthRepository). Se o login for bem-sucedido, um token JWT é gerado e retornado ao cliente para futuras autenticações. Caso contrário, é retornada uma mensagem de erro. Esse processo é crucial para garantir a segurança da aplicação, permitindo que somente usuários autenticados acessem recursos protegidos. Conforme o Anexo 7 (BackEnd).

Método Principal: **Login(LoginDTO model):** Responsável pelo processo de autenticação do usuário. Primeiramente, ele busca o usuário no banco de dados usando o **UserManager** com base no email fornecido. Em seguida, valida a senha utilizando o método **IsValidPassword**, que garante que a senha atenda aos requisitos de segurança, como conter uma letra maiúscula, um número e um símbolo especial. Após isso, o **SignInManager** é utilizado para verificar se a senha fornecida está correta. Se todas as condições forem atendidas, o método gera um **JWT** chamando o método **GenerateJwtToken**, que cria um token de autenticação. O método

então retorna uma resposta contendo o token JWT e os dados do usuário, como ID, perfil e status. Conforme o Anexo 7.1(BackEnd)

Método Secundário: GenerateJwtToken(Usuario user) : responsável por gerar um token JWT para autenticação do usuário. Ele começa recuperando as configurações de JWT do arquivo appsettings.json, como a chave secreta, tempo de expiração, audiência e emissor. Em seguida, cria as **reivindicações (claims)**, que são os dados do usuário, como email e nome, que serão incluídos no token. O token é configurado com uma data de expiração e uma assinatura utilizando o algoritmo **HMAC-SHA256** com a chave secreta. Por fim, o token é gerado e retornado como uma string. Conforme o Anexo 7.1.1 (BackEnd)

Método Secundário: IsValidPassword(string senha) : valida a complexidade da senha do usuário. Ele verifica se a senha contém, ao menos, uma letra maiúscula, um número e um símbolo especial. Se todos esses requisitos forem atendidos, o método retorna **true**; caso contrário, retorna **false**, garantindo que a senha atenda aos critérios de segurança definidos. Conforme o Anexo 7.1.1 (BackEnd)

Repositório da Consulta

O repositório de consultas contém a lógica de acesso ao banco de dados para as operações **CRUD** (Criar, Ler, Atualizar, Excluir) relacionadas às consultas. Ele inclui funções como AgendarConsulta, que permite agendar uma nova consulta para um paciente, e GetConsultasConfirmadasByPaciente, que recupera todas as consultas confirmadas de um paciente específico. Essas funções são fundamentais para gerenciar o agendamento e o acompanhamento das consultas médicas na aplicação, proporcionando uma interação eficiente com os dados relacionados às consultas no sistema. Conforme o Anexo 8 (BackEnd).

GETCONSULTAS: O método GetConsultas é responsável por buscar todas as consultas armazenadas no banco de dados e retornar seus detalhes. Ele começa recuperando as consultas utilizando o **Entity Framework**, incluindo os dados dos pacientes e médicos associados. Caso não haja consultas encontradas, ele retorna uma resposta com falha. Se consultas forem encontradas, ele as converte em uma lista de objetos **ConsultaSaidaDTO**, que inclui informações detalhadas sobre cada consulta, como o ID, data e hora, status, local, encaminhamentos, e dados completos do paciente e do médico, organizados em **DTOs** específicos para cada um. O método retorna essas informações em um formato estruturado. Se ocorrer algum erro durante o processo, ele registra o erro e retorna uma resposta de falha com uma mensagem genérica.

GetConsultasByID: O método GetConsultaById é responsável por buscar uma consulta específica no banco de dados com base no **ID** fornecido. Inicialmente, ele tenta localizar a consulta, incluindo os dados do paciente e do médico associados, utilizando o **Entity Framework**. Se a consulta não for encontrada, retorna uma resposta de falha. Caso contrário, ele mapeia os dados da consulta para um **DTO (ConsultaSaidaDTO)**, que inclui informações detalhadas sobre a consulta, o paciente e o médico, como a data e hora, status, local, encaminhamentos, e os dados pessoais completos do paciente e do médico. Se o processo ocorrer com sucesso, ele retorna os dados estruturados; se ocorrer algum erro durante a execução, ele captura a exceção, registra o erro e retorna uma mensagem de falha para o usuário.

AgendarConsulta: O método Agendar Consulta tem a finalidade de agendar uma nova consulta para um paciente com um médico específico. Primeiramente, ele realiza uma série de verificações para garantir que a data da consulta seja no futuro, que o médico e o paciente existam no sistema, verificando os IDs fornecidos. Caso alguma dessas condições não seja atendida, ele retorna uma resposta de falha com a mensagem apropriada. Se todas as validações passarem, o método cria uma nova instância de Consulta com as informações fornecidas no AgendamentoConsultaDTO, como a data, hora, local, status e os IDs do paciente e do médico. A consulta é então adicionada ao banco de dados e as mudanças são salvas. Após isso, o método tenta confirmar a consulta chamando o método ConfirmarConsulta do repositório de médicos. Caso a confirmação seja bem-sucedida, a resposta inclui uma mensagem positiva. Se houver problemas ao confirmar a consulta ou algum erro interno, a mensagem será ajustada para refletir o problema ocorrido, mas ainda retornando à consulta agendada. Se ocorrer algum erro inesperado durante o processo de agendamento, ele é tratado e registrado, retornando uma mensagem genérica de falha. Ao final, o método retorna a resposta, seja de sucesso ou de falha, com o estado do agendamento da consulta.

GetConsultasConfirmadasByPaciente: O método GetConsultasConfirmadasByPaciente busca as consultas confirmadas de um paciente, verificando inicialmente se o paciente existe no banco de dados. Caso contrário, retorna um erro informando que o paciente não foi encontrado. Se o paciente for localizado, o método consulta as consultas com o status **Confirmada**, associando os dados do médico, mas sem informações sobre o paciente. Em seguida, retorna uma lista das consultas confirmadas ou, se não houver resultados, uma mensagem informando a ausência de consultas. Em caso de erro, o método registra a falha e retorna uma mensagem genérica de erro.

DeleteConsulta: O método DeleteConsulta é responsável por excluir uma consulta do banco de dados. Ele começa verificando se a consulta com o ID fornecido existe. Se não for encontrada, retorna uma resposta informando que a consulta não foi encontrada. Caso contrário, tenta remover a consulta e salvar as alterações no banco de dados. Se a remoção for bem-sucedida, o método retorna uma mensagem de sucesso. Se ocorrer algum erro durante a exclusão, ou se a consulta não for

removida corretamente, o método captura a exceção, registra o erro e retorna uma resposta indicando falha.

Repositório de Documentos

O repositório de documentos é responsável pela manipulação e acesso a arquivos e documentos relacionados aos pacientes. Ele contém funções como SaveFileAsync, que permite salvar arquivos enviados, como exames ou receitas; GetDocumento, que recupera documentos específicos de um paciente; GetExames, que retorna os exames registrados; GetReceitas, que acessa as receitas médicas vinculadas aos pacientes; e GetOutros, que lida com outros tipos de documentos. Essas funções garantem que os documentos médicos sejam armazenados e recuperados eficientemente, promovendo uma gestão organizada e segura das informações dos pacientes. Conforme o anexo 9.

SaveFileAsync: O método SaveFileAsync é responsável por salvar um arquivo enviado pelo usuário e associá-lo a um histórico médico do paciente. Primeiro, ele verifica se existe um histórico médico para o paciente com o ID fornecido. Se não for encontrado, retorna uma mensagem de erro. Em seguida, valida se o arquivo enviado é válido. O método cria um diretório para armazenar o arquivo, gera um nome único para o arquivo e o salva no sistema de arquivos. Depois, ele cria uma entidade Documento com as informações fornecidas, como nome, categoria, descrição, data associada e caminho do arquivo. O documento é então salvo no banco de dados. Caso ocorra algum erro durante o processo, uma exceção é capturada, registrada e uma resposta de erro é retornada. Caso contrário, uma resposta de sucesso é enviada, incluindo os detalhes do documento salvo. Conforme o anexo 9.1.

GetDocumento: O método GetDocumento busca um documento específico no banco de dados, utilizando o DocumentoId fornecido. Caso o documento não seja encontrado, ele retorna uma resposta informando que o documento não foi encontrado. Se o documento existir, ele é mapeado para um DocumentoSaidaDTO, com informações como ID, nome, categoria, descrição, data associada, data de upload, caminho do arquivo, tamanho e tipo. Caso ocorra algum erro durante o processo, o

erro é registrado no log e uma mensagem de falha é retornada, pedindo que o usuário tente novamente. Conforme o anexo 9.2.

Os métodos **GetExames**, **GetReceitas** e **GetOutros** buscam documentos específicos de um paciente de acordo com categorias como "Exames", "Receitas" ou "Outros". Eles verificam se existem documentos na categoria solicitada e retornam uma resposta informando a situação. Quando encontrados, os documentos são mapeados para um DocumentoSaidaDTO, contendo informações como nome, categoria, descrição, data associada, caminho e tipo de arquivo. Em caso de erro, o sistema registra no log e retorna uma resposta de falha, pedindo que o usuário tente novamente. Conforme o anexo 9.3.

Repositório das Doenças

O repositório de doenças é responsável por gerenciar as informações relacionadas às doenças dos pacientes. Ele inclui funções como AddDoencas, que permite adicionar novas doenças ao histórico médico de um paciente; GetDoencas, que recupera as doenças registradas para um paciente específico; e UpdateDoencas, que possibilita a atualização das informações sobre as doenças já cadastradas. Essas funções são essenciais para manter o histórico médico do paciente atualizado e garantir o acesso rápido e preciso às informações de saúde relevantes, conforme o Anexo 10.

AddDoenca: O método AddDoenca é responsável por adicionar uma nova doença ao histórico médico de um paciente. Primeiramente, ele busca o histórico médico do paciente a partir do idPaciente fornecido. Se o histórico médico não for encontrado, retorna uma resposta indicando o erro. Em seguida, valida os dados da doença recebida, como a descrição e o medicamento, garantindo que ambos não sejam nulos ou vazios. Caso contrário, ele cria uma nova instância de Doenca com os dados fornecidos e a adiciona ao histórico médico do paciente. Depois, o método salva as alterações no banco de dados e cria um DTO (DoencaSaidaDTO) com as informações da doença para retorno. Se ocorrer um erro em qualquer parte do processo, é registrado um erro no log e uma resposta de falha é retornada, solicitando ao usuário tentar novamente. O método utiliza a abordagem assíncrona, permitindo que as operações no banco de dados não bloqueiem a execução do sistema, conforme o Anexo 10.1.

GetDoencas: O método GetDoencas é responsável por recuperar todas as doenças associadas ao histórico médico de um paciente. Ele começa buscando o histórico médico do paciente, incluindo as doenças associadas, através do idPaciente fornecido. Caso o histórico médico não seja encontrado, o método retorna uma resposta de erro. Se o histórico for encontrado, as doenças são mapeadas para uma lista de DTOs (DoençaSaidaDTO), incluindo informações como o ID do histórico, o ID da doença, a descrição e o medicamento. Em seguida, a lista de doenças é retornada com sucesso. Se houver um erro durante o processo, ele é registrado no log e uma resposta de falha é retornada ao usuário, solicitando tentar novamente. O método é assíncrono, garantindo que a consulta ao banco de dados não bloqueie a execução do sistema, conforme o Anexo 10.2.

UpdateDoenca: O método UpdateDoenca é responsável por atualizar os dados de uma doença existente no banco de dados. Ele começa realizando uma verificação básica de validação, garantindo que os campos obrigatórios, como a descrição e o medicamento, não estejam vazios. Se esses dados forem inválidos, uma resposta de erro é retornada. O método então busca a doença pelo id fornecido e, se não encontrar a doença, retorna um erro informando que a doença não foi encontrada. Se a doença for localizada, seus dados (descrição e medicamento) são atualizados com os valores fornecidos no modelo. Após a atualização, as mudanças são salvas no banco de dados. O método então cria um DTO (DoencaSaidaDTO) com as informações atualizadas da doença e retorna uma resposta de sucesso. Se ocorrer um erro durante o processo, ele é registrado no log, e uma resposta de falha é retornada, sugerindo que o usuário tente novamente. O método é assíncrono, o que permite que a operação no banco de dados seja realizada de forma eficiente sem bloquear o fluxo do sistema, conforme o Anexo 10.3.

Repositório do Médico

O repositório de médicos gerência as operações relacionadas aos médicos e suas consultas. Ele inclui métodos **CRUD** (Criar, Ler, Atualizar, Excluir) para gerenciar as informações dos médicos, como dados pessoais e especialidades. A função ConfirmarConsulta permite confirmar a presença de um médico em uma consulta agendada. O método FiltrarMedico facilita a busca por médicos com base em critérios específicos, enquanto GetEspecialidade e GetLocalidades retornam, respectivamente, às especialidades médicas e as

localizações dos médicos disponíveis. Além disso, as funções auxiliares IsWeekend e IsValidDate verificam, respectivamente, se uma data é no final de semana ou se é uma data válida para agendamento. Essas funcionalidades garantem uma gestão eficiente das consultas e a precisão na disponibilidade dos médicos, conforme o anexo 11.

Repositório do Paciente

O repositório de pacientes é responsável pela gestão das informações dos pacientes no sistema. Ele inclui métodos **CRUD** (Criar, Ler, Atualizar, Excluir) para realizar operações essenciais no cadastro de pacientes, como adicionar novos registros, consultar dados existentes, atualizar informações e excluir registros de pacientes. Essas funções garantem a manutenção e integridade dos dados dos pacientes, permitindo um gerenciamento eficiente das informações pessoais e médicas dentro da aplicação. Conforme o Anexo 12.

Repositório da Vacinas

O repositório de vacinas gerencia as informações sobre as vacinas dos pacientes. Ele inclui funções como AddVacina, que permite adicionar novas vacinas ao histórico de imunização de um paciente; GetVacina, que recupera as vacinas registradas para um paciente específico; e UpdateVacina, que possibilita a atualização dos registros de vacinas já cadastradas. Essas funções são essenciais para manter o histórico de vacinação dos pacientes atualizado e garantir que as informações sobre imunizações estejam sempre acessíveis e precisas. Conforme o Anexo 13.

Considerações finais

A partir do desenvolvimento do sistema TechSaúde foi possível executar a base teórica e prática adquiridas ao longo do curso Técnico em Informática Integrado ao Ensino Médio ofertado pelo Instituto Federal de Educação, Ciência e Tecnologia do Estado de São Paulo - Campus Cubatão.

No mais, cabe destacar que o projeto proporcionou um vasto conhecimento interdisciplinar de suma importância para obtenção do título de Técnico em Informática. Sendo assim, ao longo do desenvolvimento foram alcançadas as seguintes competências: liderança, trabalho em equipe, comunicação, proatividade, criatividade, pontualidade, dentre outras habilidades.

Portanto, o sistema TechSaúde representa a ideia de cinco alunas que se propuseram a desenvolvê-lo ao longo de sete meses. O que envolveu o trabalho de muita pesquisa, estudo e comprometimento individual e coletivo. A partir da conclusão do trabalho fica eternizada a colaboração de cada membro da equipe e do Professor Orientador Mauricio Asenjo Neves.

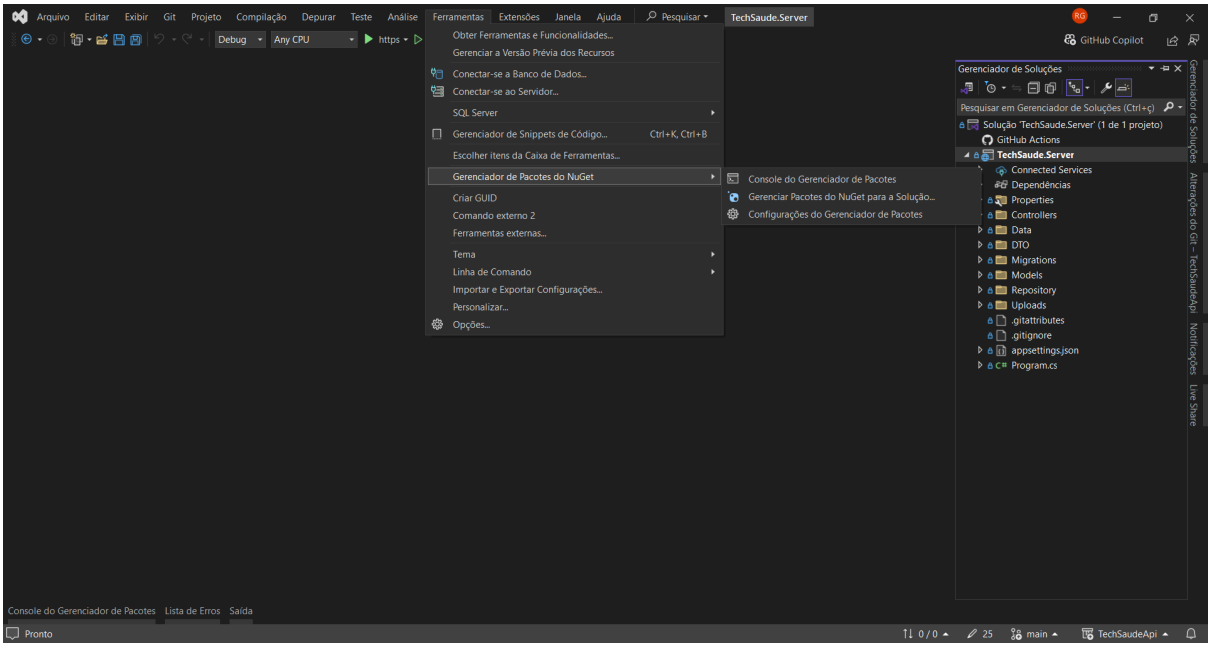
O projeto, futuramente, pode passar por uma etapa de aperfeiçoamento e implementação de recursos e atualizações em geral. No entanto, em nome do grupo, entregamos uma versão impecável e que conseguiu atingir o seu objetivo traçado inicialmente em março de 2024.

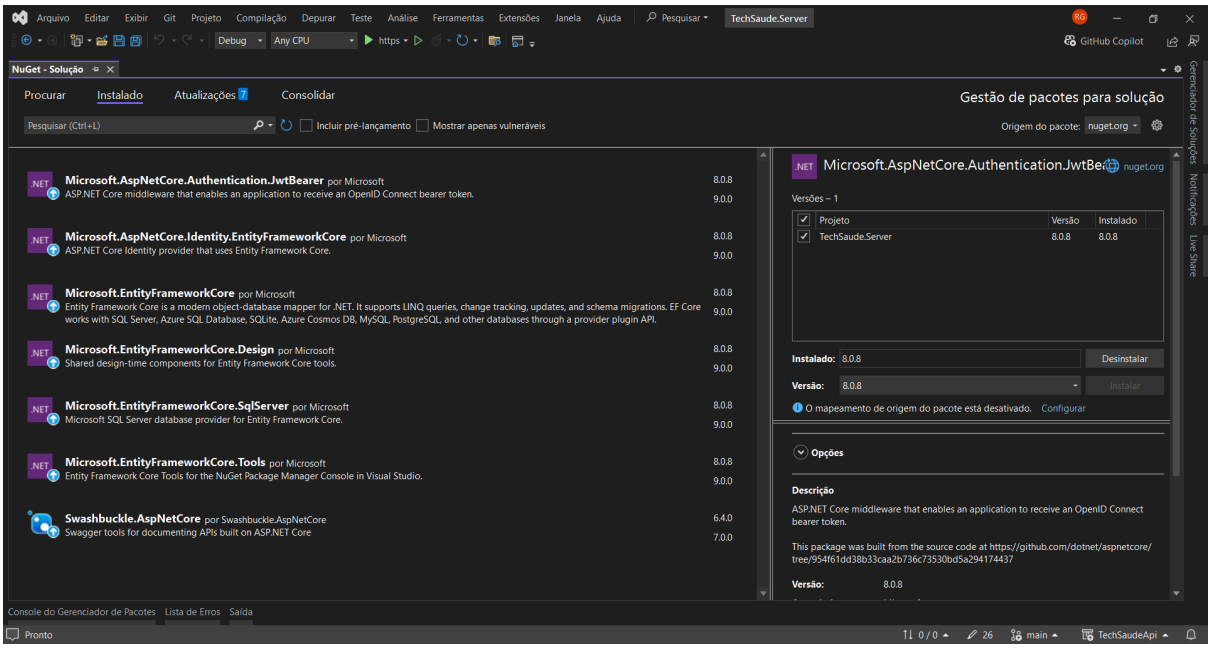
Essa apostila contém não só as etapas de desenvolvimento do sistema, mas também a nossa trajetória. Por isso, dedicamos esse trabalho às nossas famílias e aos amigos que foram grande fonte de incentivo em momentos desafiadores. A todos, o nosso mais profundo agradecimento. Em especial, agradecemos também ao Professor e Orientador Maurício Asenjo Neves, pelas correções acadêmicas, momentos de incentivo e palavras de encorajamento.

ANEXO I - Back-end:

Back-end:

Anexo 1 - BackEnd: Instalação das bibliotecas





Anexo 2 - BackEnd : Código de Conexão com o Banco de Dados

AppDbContext.cs

```
Data > AppDbContext.cs > ...
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using TechSaude.Server.Models;

namespace TechSaude.Server.Data
{
    public class AppDbContext :IdentityDbContext<Usuario, IdentityRole<int>, int>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {
        }

        public DbSet<Paciente> Pacientes { get; set; }
        public DbSet<Medico> Medicos { get; set; }
        public DbSet<Consulta> Consultas { get; set; }
        public DbSet<HistoricoMedico> HistoricosMedicos { get; set; }

        // HISTORICO MEDICO
        public DbSet<Alergia> Alergias { get; set; }
        public DbSet<Vacina> Vacinas { get; set; }
        public DbSet<Doenca> Doencas { get; set; }

        public DbSet<Documento> Documentos { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        { ...
        }
    }

}
```

appsettings.json

```
{} appsettings.json > {} Jwt
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Server=BOOK-P9HCJ2SU4O\\SQLEXPRESS;Database=DbTechSaude;Trusted_Connection=True;TrustServerCertificate=true;Connect Timeout=60;"
  },
  "Jwt": {
    "SecretKey": "dbbbd1be042ba2c5b6c757c396dffe99f202855f84c682ce0fb2a68aa0ccd01a",
    "Issuer": "http://localhost:7021",
    "Audience": "TechSaude",
    "ExpiryMinutes": 60
  },
  "AllowedHosts": "*"
}
```

Anexo 3 - BackEnd : Código do Program.cs

Program.cs M

Program.cs > ...

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using System.Text;
using System.Text.Json.Serialization;
using TechSaude.Server.Data;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.AlergiasRepository;
using TechSaude.Server.Repository.AuthRepository;
using TechSaude.Server.Repository.ConsultaRespository;
using TechSaude.Server.Repository.DocumentosRepository;
using TechSaude.Server.Repository.DoencasRepository;
using TechSaude.Server.Repository.MedicoRepository;
using TechSaude.Server.Repository.PacienteRepository;
using TechSaude.Server.Repository.VacinaRepository;

var builder = WebApplication.CreateBuilder(args);
// Configuracao do DbContext
builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")!));
// Configuracao dos Repositories
builder.Services.AddScoped<IAuthRepository, AuthRepository>();
builder.Services.AddScoped<IMedicoRepository, MedicoRepository>();
builder.Services.AddScoped<IPacienteRepository, PacienteRepository>();
builder.Services.AddScoped<IConsultaRepository, ConsultaRepository>();
builder.Services.AddScoped<IVacinaRepository, VacinasRepository>();
builder.Services.AddScoped<IAlergiasRepository, AlergiasRepository>();
builder.Services.AddScoped<IDoencasRepository, DoencasRepository>();
builder.Services.AddScoped<IDocumentoRepository, DocumentoRepository>();
// Configuracao do Identity
builder.Services.AddIdentity<Usuario, IdentityRole<int>>(options =>
{
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireUppercase = true;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
})
.AddEntityFrameworkStores<AppDbContext>()
.AddDefaultTokenProviders();
```

Program.cs M

Program.cs > ...

```

// Configuracao do JWT
var jwtSettings = builder.Configuration.GetSection("Jwt");
var jwtSecretKey = jwtSettings["SecretKey"];
var jwtExpiryMinutes = int.TryParse(jwtSettings["ExpiryMinutes"], out var expiry) ? expiry : 60; // Default value
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        ValidAudience = builder.Configuration["Jwt:Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(jwtSecretKey ?? string.Empty))
    };
});
// Configuracao do CORS
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowFrontend", policy =>
    {
        policy.WithOrigins("http://localhost:5173") // Permitir apenas o frontend
              .AllowAnyMethod() // Permitir qualquer m◆todo HTTP (GET, POST, etc.)
              .AllowAnyHeader(); // Permitir qualquer cabe◆alho
    });
});
// Configuracao dos Controllers e JSON
builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.Preserve;
    });
```

Program.cs M

Program.cs > ...

```
82
83  // Configuracao do Swagger/OpenAPI
84  builder.Services.AddEndpointsApiExplorer();
85  builder.Services.AddSwaggerGen(c =>
86  {
87      c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
88      {
89          Name = "Authorization",
90          Type = SecuritySchemeType.Http,
91          Scheme = "Bearer",
92          BearerFormat = "JWT",
93          In = ParameterLocation.Header,
94          Description = "Insira o token JWT no campo abaixo.\nExemplo: 'Bearer {token}'"
95      });
96      c.AddSecurityRequirement(new OpenApiSecurityRequirement { {
97              new OpenApiSecurityScheme
98              {
99                  Reference = new OpenApiReference
100                 {
101                     Type = ReferenceType.SecurityScheme, Id = "Bearer"
102                 }
103             },  new string[] {}   }
104     });
105 });
106 var app = builder.Build();
107 // Configuracao do pipeline de requisi��es HTTP
108 if (app.Environment.IsDevelopment()){ app.UseSwagger();
109     app.UseSwaggerUI(c => {c.SwaggerEndpoint("/swagger/v1/swagger.json", "TechSaude API V1"); }); }
110
111 app.UseHttpsRedirection();
112 app.UseCors("AllowFrontend");
113
114 app.UseAuthentication();
115 app.UseAuthorization();
116
117 app.MapControllers();
118 app.Run();
```

Anexo 4 - BackEnd : Códigos dos Modelos

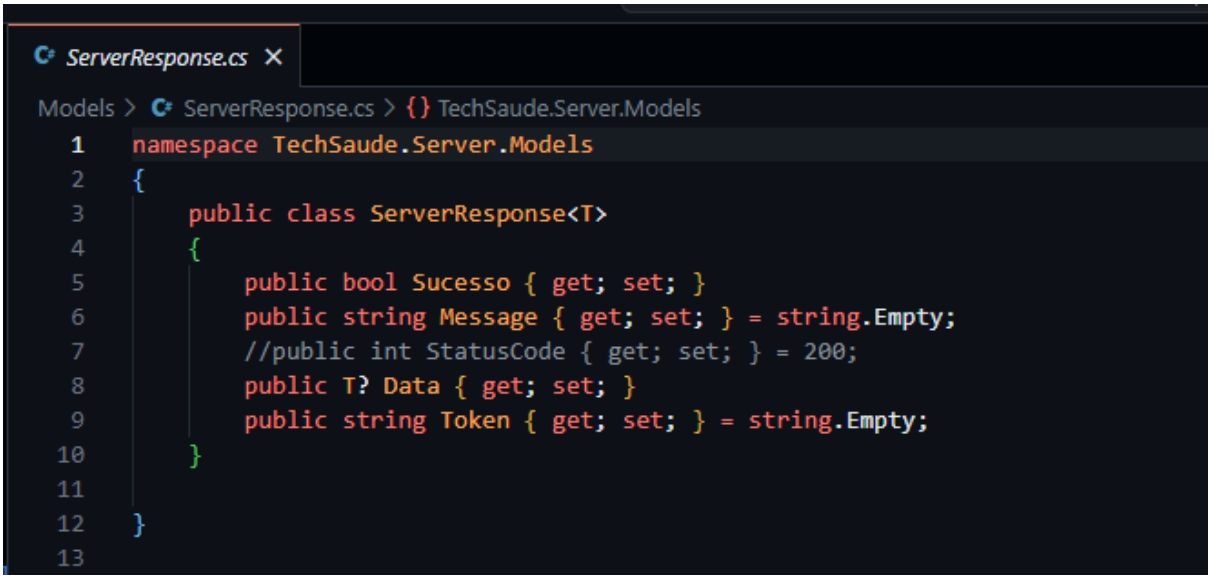
Modelo do Usuário

Usuario.cs

Models > Usuario.cs > {} TechSaude.Server.Models > Usuario

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;

namespace TechSaude.Server.Models
{
    public class Usuario : IdentityUser<int>
    {

        public DateTime  DataCadastro { get; set; } = DateTime.Now;
        public StatusUserEnum Status { get; set; }
        public PerfilUserEnum PerfilUser { get; set; }
        public bool Termo { get; set; }
        public bool Compartilhamento { get; set; }

    }
    [JsonConverter(typeof(JsonStringEnumConverter))]
    public enum StatusUserEnum
    {
        Ativo = 1,
        Inativo = 2,
        Bloqueado = 3
    }
    [JsonConverter(typeof(JsonStringEnumConverter))]
    public enum PerfilUserEnum
    {
        Administrador,
       Paciente
    }


}

```

Modelo do Server Response



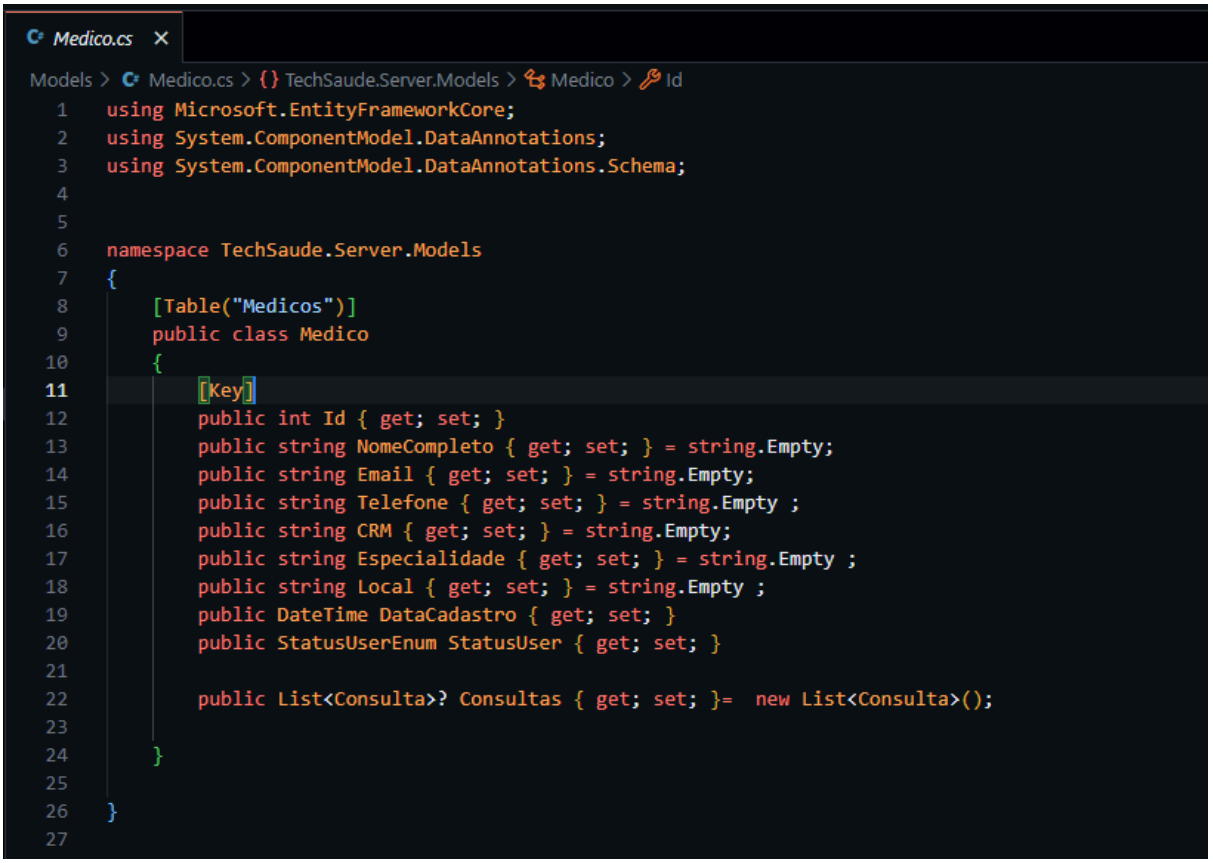
Modelo do Paciente

Paciente.cs

Models > Paciente.cs > ...

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json.Serialization;

namespace TechSaude.Server.Models
{
    [Table("Pacientes")]
    public class Paciente : Usuario
    {

        public string NomeCompleto { get; set; } = string.Empty;
        public DateTime DataNascimento { get; set; }
        public SexoEnum Sexo { get; set; }
        public string Endereco { get; set; } = string.Empty;
        public string Telefone { get; set; } = string.Empty;
        public string CNS { get; set; } = string.Empty;
        public string Convenio { get; set; } = string.Empty;

        public List<Consulta> Consultas { get; set; } = new List<Consulta>();
        public HistoricoMedico? HistoricosMedicos { get; set; }

    }
    [JsonConverter(typeof(JsonStringEnumConverter))]
    public enum SexoEnum
    {
        Feminino,
        Masculino,
        Outro,
        Vazio
    }
}

```

Modelo do Médico



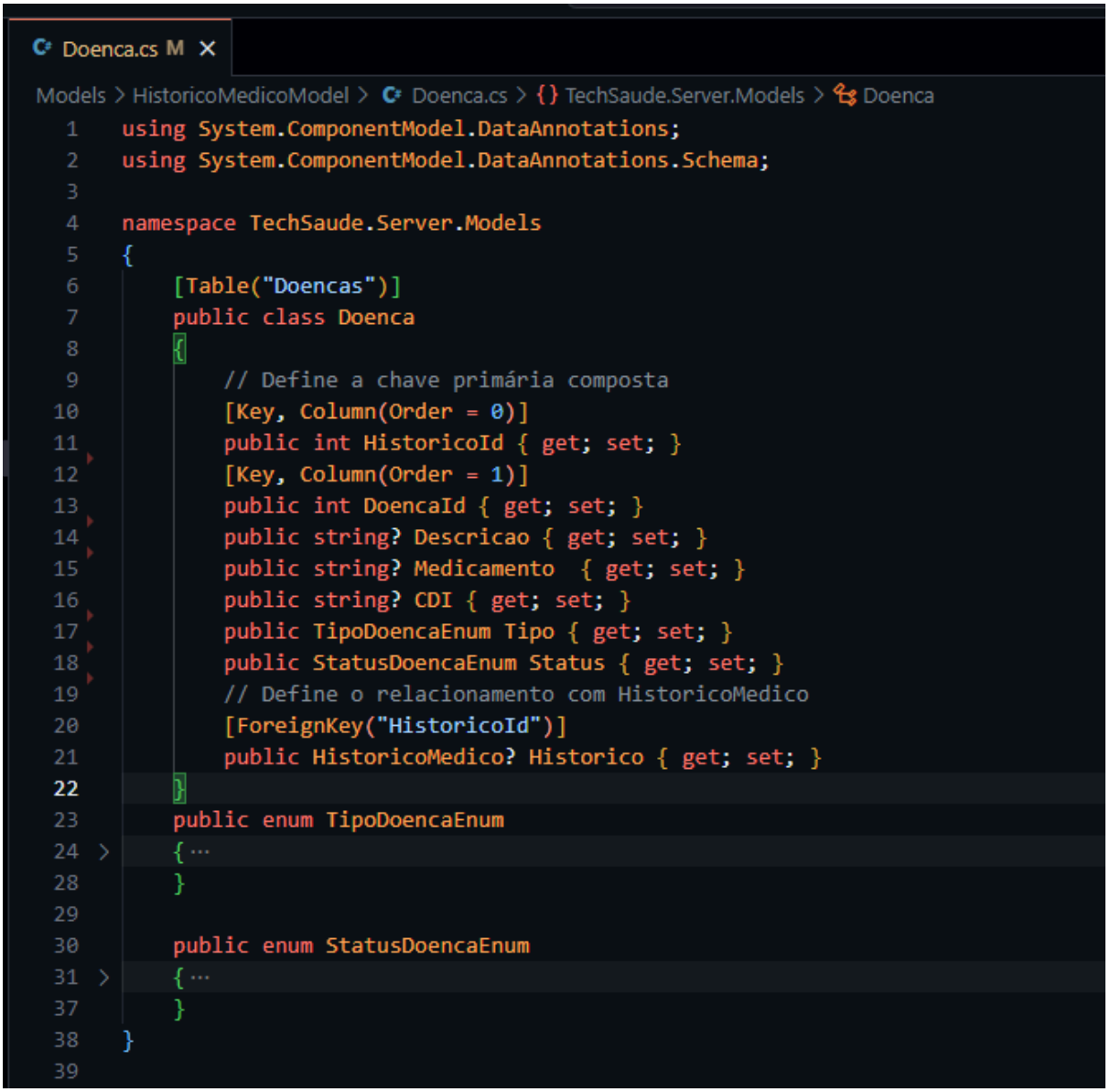
Modelo do Histórico Médico

HistoricoMedico.cs

Models > HistoricoMedico.cs > ...

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace TechSaude.Server.Models
{
    [Table("HistoricoMedicos")]
    public class HistoricoMedico
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public int PacienteId { get; set; }
        [ForeignKey("PacienteId")]
        public virtual Paciente? Paciente { get; set; }

        public DateTime DataRegistro { get; set; } = DateTime.UtcNow;
        public TipoSanguineoEnum TipoSanguineo { get; set; }


        public virtual List<Documento>? Documentos { get; set; } = new List<Documento>();

        public virtual List<Alergia>? Alergias { get; set; } = new List<Alergia>();
        public virtual List<Vacina>? Vacinas { get; set; } = new List<Vacina>();
        public virtual List<Doenca>? Doencas { get; set; } = new List<Doenca>();


    }

    public enum TipoSanguineoEnum
    {
        A_positivo,
        A_negativo,
        B_positivo,
        B_negativo,
        AB_positivo,
        AB_negativo,
        O_positivo,
        O_negativo,
        NaoInformado
    }
}

```

Modelo da Doença



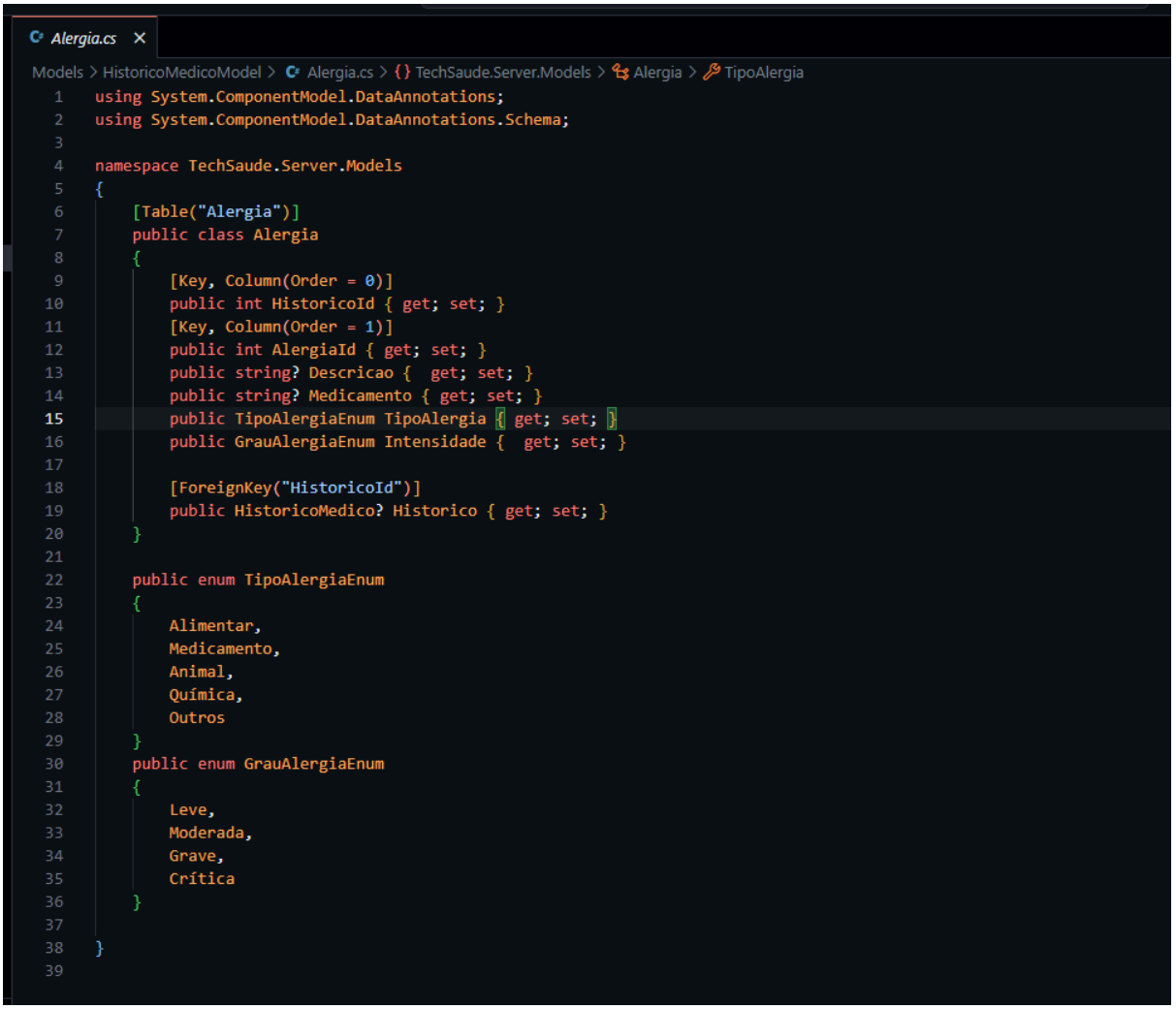
Modelo da Consulta

Consulta.cs

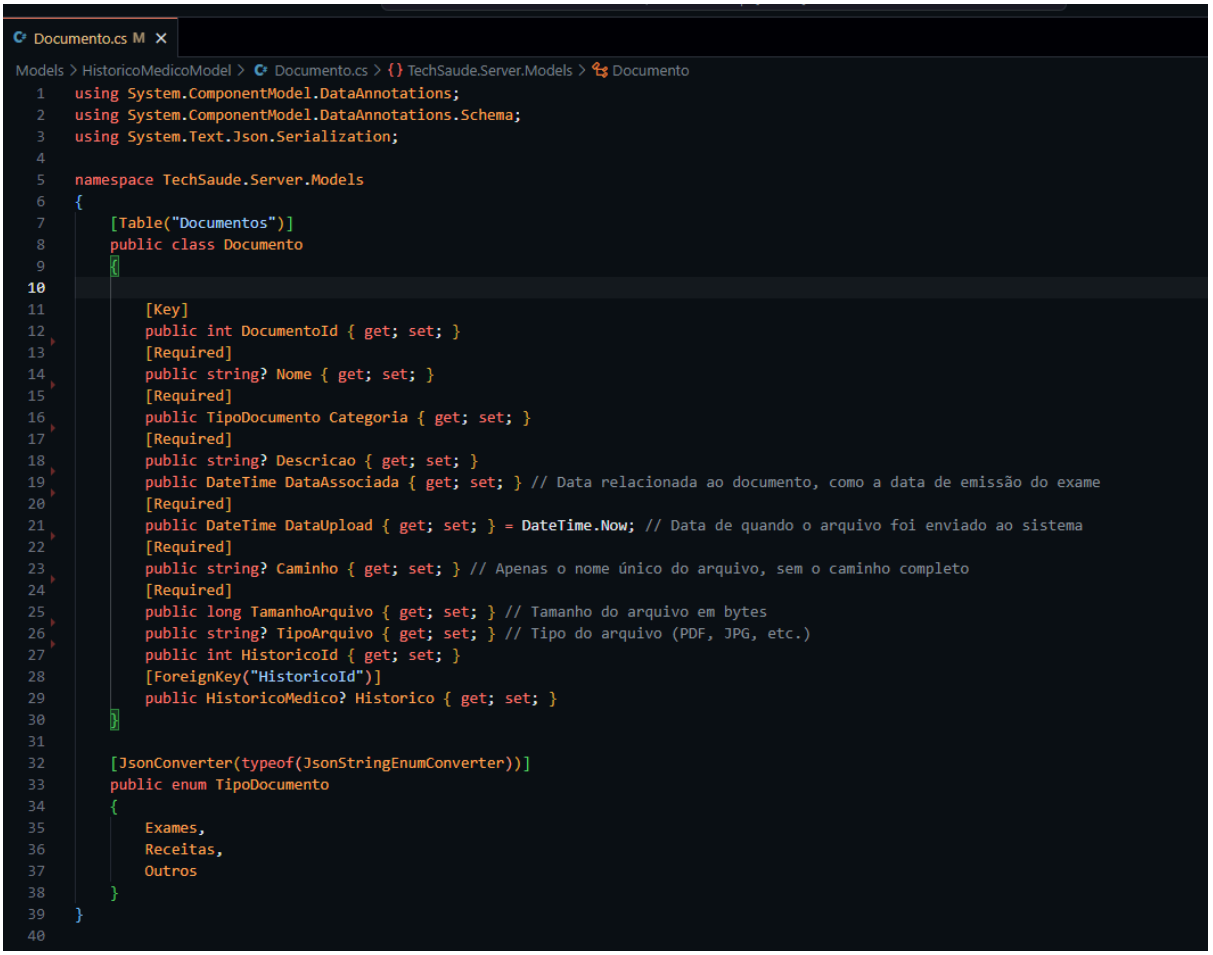
Models > Consulta.cs > { } TechSaude.Server.Models > Consulta

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json.Serialization;

namespace TechSaude.Server.Models
{
    [Table("Consultas")]
    public class Consulta
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public int PacienteId { get; set; }
        [Required]
        public int MedicoId { get;set; }

        public DateTime DataHora { get; set; }
        public StatusConsultaEnum Status { get; set; }
        public string? Local {  get; set; }
        public string? Motivo { get; set; }

        public string? Encaminhamentos { get; set; } // Arquivo

        // Relacionamentos
        [ForeignKey(nameof(PacienteId))]
        public virtual Paciente? Paciente { get; set; }

        [ForeignKey(nameof(MedicoId))]
        public virtual Medico? Medico { get; set; }
    }

    [JsonConverter(typeof(JsonStringEnumConverter))]
    public enum StatusConsultaEnum
    {
        Agendada,
        Confirmada,
        Cancelada,
        Realizada
    }
}

```

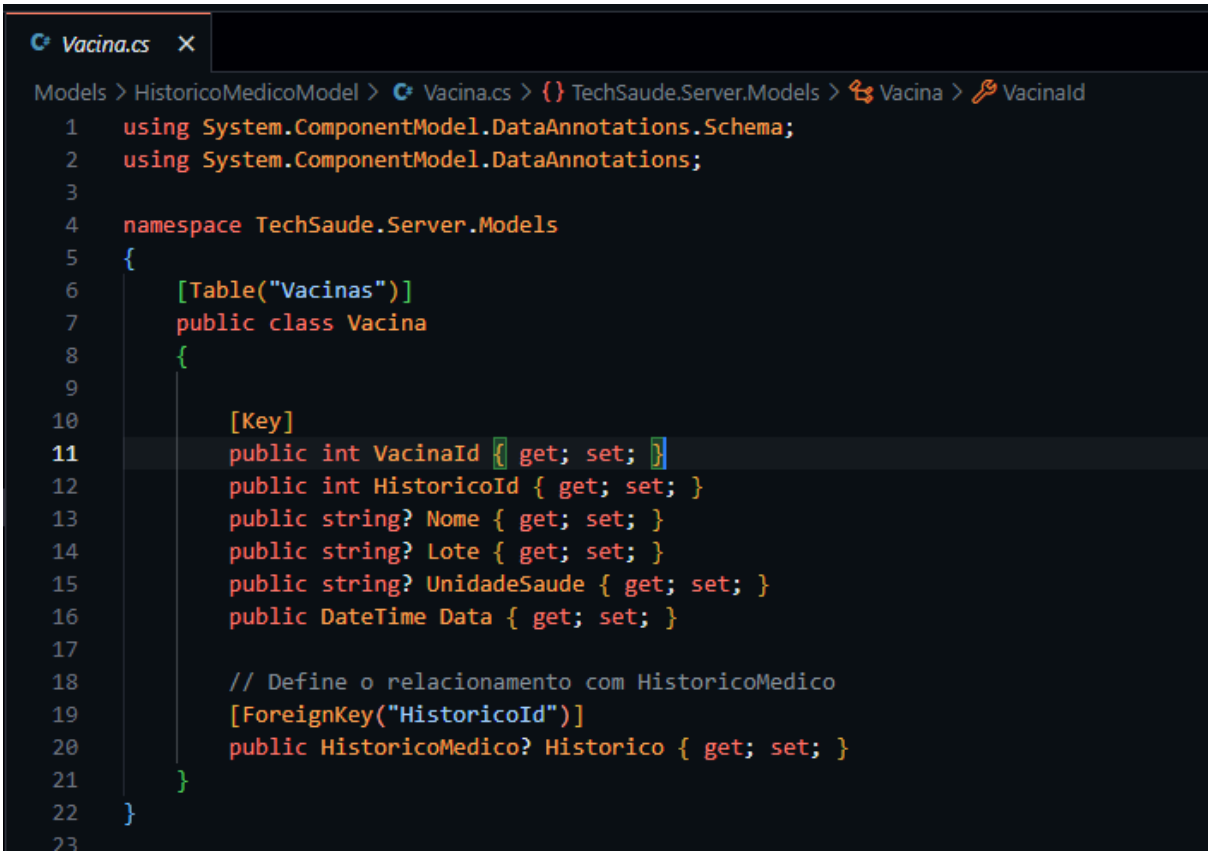
Modelo da Alergia



Modelo do Documento



Modelo da Vacina



Anexo 5 - BackEnd : Código dos Controllers

Paciente

PacientesController.cs M

Controllers > PacientesController.cs > { } TechSaude.Server.Controllers > PacientesController

```cs
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.PacienteRepository;

namespace TechSaude.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PacientesController : ControllerBase
    {
        private readonly IPacienteRepository _pacienteRepository;

        public PacientesController(IPacienteRepository pacienteRepository){
            _pacienteRepository = pacienteRepository;
        }
        [HttpGet]
        public async Task<ServerResponse<List<UsuarioSaidaDTO>>> GetPacientes() {
            return await _pacienteRepository.GetPacientes();
        }
        [HttpGet]
        [Route("{pacienteId}")]
        public async Task<ServerResponse<UsuarioSaidaDTO>> GetPacienteById(int pacienteId) {
            return await _pacienteRepository.GetPacienteById(pacienteId);
        }
        [HttpPost]
        public async Task<ServerResponse<UsuarioSaidaDTO>> AddPaciente([FromBody] RegisterPacienteDTO model){
            return await _pacienteRepository.CreatePaciente(model);
        }
        [HttpPut]
        [Route("{pacienteId}")]
        public async Task<ServerResponse<UsuarioSaidaDTO>> UpdatePaciente(int pacienteId, PerfilPacienteDTO model){
            return await _pacienteRepository.UpdatePaciente(pacienteId, model);
        }
        [HttpDelete]
        [Route("{pacienteId}")]
        public async Task<ServerResponse<bool>> DeleteMedico(int pacienteId){
            return await _pacienteRepository.DeletePaciente(pacienteId);
        }
    }
}
```

Médicos

MedicosController.cs M

Controllers > MedicosController.cs > ...

```cs
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.MedicoRepository;

namespace TechSaude.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MedicosController : ControllerBase
    {
        private readonly IMedicoRepository _medicoRepository;
        public MedicosController(IMedicoRepository medicoRepository){
            _medicoRepository = medicoRepository;
        }
        // Endpoint para obter todos os médicos
        [HttpGet]
        public async Task<ActionResult<ServerResponse<List<MedicoDTO>>>> GetMedicos() {
            var response = await _medicoRepository.GetMedicos();
            if (response.Sucesso) return Ok(response);
            return BadRequest(response);
        }
        // Endpoint para obter um médico pelo ID
        [HttpGet("{medicoId}")]
        public async Task<ActionResult<ServerResponse<MedicoDTO>>> GetMedicoById([FromRoute] int medicoId){
            var response = await _medicoRepository.GetMedicoById(medicoId);
            if (response.Sucesso) return Ok(response);
            return NotFound(response);
        }
        // Endpoint para registrar um novo médico
        [HttpPost]
        public async Task<ActionResult<ServerResponse<MedicoDTO>>> Register([FromBody] RegisterMedicoDTO model)
        {
            var response = await _medicoRepository.CreateMedico(model);
            if (response.Sucesso)  return CreatedAtAction(nameof(GetMedicoById), new { medicoId = response.Data!.Id }, response);
            return BadRequest(response);
        }
        // Endpoint para atualizar um médico existente
        [HttpPut("{medicoId}")]
        public async Task<ActionResult<ServerResponse<MedicoDTO>>> UpdateMedico([FromRoute] int medicoId, [FromBody] Medico model) {
            var response = await _medicoRepository.UpdateMedico(medicoId, model);
            if (response.Sucesso)  return Ok(response);
            return NotFound(response);
        }
        // Endpoint para deletar um médico
        [HttpDelete("{medicoId}")]
        public async Task<ActionResult<ServerResponse<bool>>> DeleteMedico([FromRoute] int medicoId){
            var response = await _medicoRepository.DeleteMedico(medicoId);
            if (response.Sucesso) return Ok(response);
            return NotFound(response);
        }
        [HttpGet("especialidades")]
        public async Task<ActionResult<ServerResponse<List<EspecialidadeDTO>>>> GetEspecialidades() {
            var response = await _medicoRepository.GetEspecialidades();
            if (response.Sucesso) return Ok(response);
            return BadRequest(response);
        }
        [HttpGet("localidades")]
        public async Task<ActionResult<ServerResponse<List<LocalidadeDTO>>>> GetLocalidades([FromQuery] string especialidade) {
            var response = await _medicoRepository.GetLocalidades(especialidade);
            if (response.Sucesso) return Ok(response);
            return BadRequest(response);
        }
        [HttpGet("filtrar")]
        public async Task<ActionResult<ServerResponse<List<MedicoDTO>>>> FiltrarMedicos([FromQuery] string especialidade, [FromQuery] string localidade){
            var response = await _medicoRepository.FiltrarMedicos(especialidade, localidade);
            if (response.Sucesso)return Ok(response);
            return BadRequest(response);
        }
    }
}
```

Consultas

ConsultasController.cs M

Controllers > ConsultasController.cs > ...

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.ConsultaRespository;

namespace TechSaude.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ConsultasController : ControllerBase
    {
        private readonly AppDbContext _context;
        private readonly IConsultaRepository _consultaRepository;

        public ConsultasController(AppDbContext context, IConsultaRepository consultaRepository){
            _context = context;
            _consultaRepository = consultaRepository;
        }
        [HttpGet]
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultas(){
            return await _consultaRepository.GetConsultas();
        }
        [HttpGet("{consultaId}")]
        public async Task<ServerResponse<ConsultaSaidaDTO>> GetConsultaById(int consultaId){
            return await _consultaRepository.GetConsultaById(consultaId);
        }
        [HttpGet("medico/{medicoId}")]
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultas([FromRoute] int medicoId){
            return await _consultaRepository.GetConsultasByMedico(medicoId);
        }
        [HttpDelete]
        public async Task<ServerResponse<bool>> DeleteConsulta(int consultaId){
            return await _consultaRepository.DeleteConsulta(consultaId);
        }
        [HttpPost("agendamento/{pacienteId}")]
        public async Task<ServerResponse<Consulta>> AgendamentoConsulta(int pacienteId, [FromBody] AgendamentoConsultaDTO agendamentoConsulta){
            return await _consultaRepository.AgendarConsulta(pacienteId, agendamentoConsulta);
        }
        [HttpGet("paciente/{pacienteId}")]
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> ConsultasPaciente(int pacienteId){
            return await _consultaRepository.GetConsultasByPaciente(pacienteId);
        }
        [HttpGet("confirmadas")]
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> ConsultasConfirmadasPaciente(int pacienteId){
            return await _consultaRepository.GetConsultasConfirmadasByPaciente(pacienteId);
        }
    }
}
```

Auth

AuthController.cs

Controllers > AuthController.cs > ...

```
using Microsoft.AspNetCore.Mvc;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.AuthRepository;

namespace TechSaude.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthRepository _authRepository;


        public AuthController(IAuthRepository authRepository)
        {
            _authRepository = authRepository;
        }

        [HttpPost("login")]

        public async Task<ServerResponse<ResponseLoginDTO>> Login([FromBody] LoginDTO model)
        {
            return await _authRepository.Login(model);
        }

    }
}
```

Vacinas

VacinasController.cs M

Controllers > HistoricoMedico > VacinasController.cs > {} TechSaude.Server.Controllers.HistoricoMedico > VacinasController

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore.Metadata.Internal;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.VacinaRepository;

namespace TechSaude.Server.Controllers.HistoricoMedico
{
    [Route("api/[controller]")]
    [ApiController]
    public class VacinasController : ControllerBase
    {
        public readonly IVacinaRepository _vacinaRepository;

        public VacinasController(IVacinaRepository vacina)
        {
            _vacinaRepository = vacina;
        }
        [HttpGet("{idPaciente}")]
        public async Task<ServerResponse<List<VacinaSaidaDTO>>> GetVacinasByPaciente(int idPaciente)
        {
            return await _vacinaRepository.GetVacinas(idPaciente);
        }
        [HttpPost("{idPaciente}")]
        public async Task<ServerResponse<VacinaSaidaDTO>> AddVacinasByPaciente(int idPaciente, [FromBody] VacinaDTO model)
        {
            return await _vacinaRepository.AddVacina(idPaciente, model);
        }
        [HttpPut("{idVacina}")]
        public async Task<ServerResponse<VacinaSaidaDTO>> UpdateVacinaByPaciente(int idVacina, [FromBody] VacinaDTO model)
        {
            return await _vacinaRepository.UpdateVacina(idVacina, model);
        }

    }
}
```

Doenças

DoencasController.cs M

Controllers > HistoricoMedico > DoencasController.cs > ...

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore.Metadata.Internal;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.DoencasRepository;
using TechSaude.Server.Repository.VacinaRepository;

namespace TechSaude.Server.Controllers.HistoricoMedico
{
    [Route("api/[controller]")]
    [ApiController]
    public class DoencasController : ControllerBase
    {
        public readonly IDoencasRepository _doencaRepository;

        public DoencasController(IDoencasRepository doenca)
        {
            _doencaRepository = doenca;
        }
        [HttpGet("{idPaciente}")]
        public async Task<ServerResponse<List<DoencaSaidaDTO>>> GetVacinasByPaciente(int idPaciente)
        {
            return await _doencaRepository.GetDoencas(idPaciente);
        }
        [HttpPost("{idPaciente}")]
        public async Task<ServerResponse<DoencaSaidaDTO>> AddVacinasByPaciente(int idPaciente, [FromBody] DoencasDTO model)
        {
            return await _doencaRepository.AddDoenca(idPaciente, model);
        }
        [HttpPut]
        public async Task<ServerResponse<DoencaSaidaDTO>> UpdateVacinaByPaciente(int idDoenca, [FromBody] DoencasDTO model)
        {
            return await _doencaRepository.UpdateDoenca(idDoenca, model);
        }

    }
}
```

Documento

DocumentoController.cs M

Controllers > HistoricoMedico > DocumentoController.cs > ...

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.DocumentosRepository;

namespace TechSaude.Server.Controllers.HistoricoMedico
{
    [Route("api/[controller]")]
    [ApiController]
    public class DocumentoController : ControllerBase
    {
        private readonly IDocumentoRepository _fileService;
        private readonly AppDbContext _context;
        public DocumentoController(IDocumentoRepository fileService, AppDbContext context){
            _fileService = fileService;
            _context = context;
        }
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = ""){ return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };}
        [HttpPost("upload")]
        public async Task<ServerResponse<DocumentoSaidaDTO>> Upload(FileDTO model){
          return await _fileService.SaveFileAsync(model);
        }
        [HttpGet("{id}")]
        public async Task<ServerResponse<DocumentoSaidaDTO>> GetDocumento(int id){
            return await _fileService.GetDocumento(id);
        }
        [HttpGet("exames")]
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetExames(int pacienteId)  {
            return await _fileService.GetExames(pacienteId);
        }
        [HttpGet("receitas")] public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetReceitas(int pacienteId)
        {
            return await _fileService.GetReceitas(pacienteId);
        }
        [HttpGet("outros")] public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetOutrosDoc(int pacienteId)
        {
            return await _fileService.GetOutros(pacienteId);
        }
```

DocumentoController.cs M

Controllers > HistoricoMedico > DocumentoController.cs > ...

```
namespace TechSaude.Server.Controllers.HistoricoMedico
    public class DocumentoController : ControllerBase
        }
        [HttpGet("download/{id}")]
        public async Task<IActionResult> Download(int id,  [FromServices] IWebHostEnvironment env)
        {
            var documento = await _context.Documentos.FindAsync(id);
            if (documento == null)
                return NotFound(new { message = "Documento não encontrado." });

            var uploadsPath = Path.Combine(env.ContentRootPath, "Uploads");
            var filePath = Path.Combine(uploadsPath, documento.Caminho!);

            if (!System.IO.File.Exists(filePath))
                return NotFound(new { message = "Arquivo não encontrado no servidor." });

            var memory = new MemoryStream();
            using (var stream = new FileStream(filePath, FileMode.Open))
            {
                await stream.CopyToAsync(memory);
            }
            memory.Position = 0;

            return File(memory, documento.TipoArquivo ?? "application/octet-stream", documento.Nome);
        }
    }
}
```

Alergias

AlergiasController.cs

Controllers > HistoricoMedico > AlergiasController.cs > ...

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore.Metadata.Internal;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.AlergiasRepository;
using TechSaude.Server.Repository.VacinaRepository;

namespace TechSaude.Server.Controllers.HistoricoMedico
{
    [Route("api/[controller]")]
    [ApiController]
    public class AlergiasController : ControllerBase
    {
        public readonly IAlergiasRepository _alergiaRepository;

        public AlergiasController(IAlergiasRepository alergia)
        {
            _alergiaRepository = alergia;
        }
        //[Authorize]
        [HttpGet("{idPaciente}")]
        public async Task<ServerResponse<List<AlergiaSaidaDTO>>> GetAlergiasByPaciente(int idPaciente)
        {
            return await _alergiaRepository.GetAlergias(idPaciente);
        }
        //[Authorize]
        [HttpPost("{idPaciente}")]
        public async Task<ServerResponse<AlergiaSaidaDTO>> AddAlergiaByPaciente(int idPaciente, [FromBody] AlergiasDTO model)
        {
            return await _alergiaRepository.AddAlergia(idPaciente, model);
        }
        //[Authorize]
        [HttpPut("{idAlergia}")]
        public async Task<ServerResponse<AlergiaSaidaDTO>> UpdateAlergiaByPaciente(int idAlergia, [FromBody] AlergiasDTO model)
        {
            return await _alergiaRepository.UpdateAlergia(idAlergia, model);
        }

    }
}
```

Anexo 6 - BackEnd : Código do Repositório das Alergias

AlergiasRepository.cs M

Repository > AlergiasRepository > AlergiasRepository.cs > ...

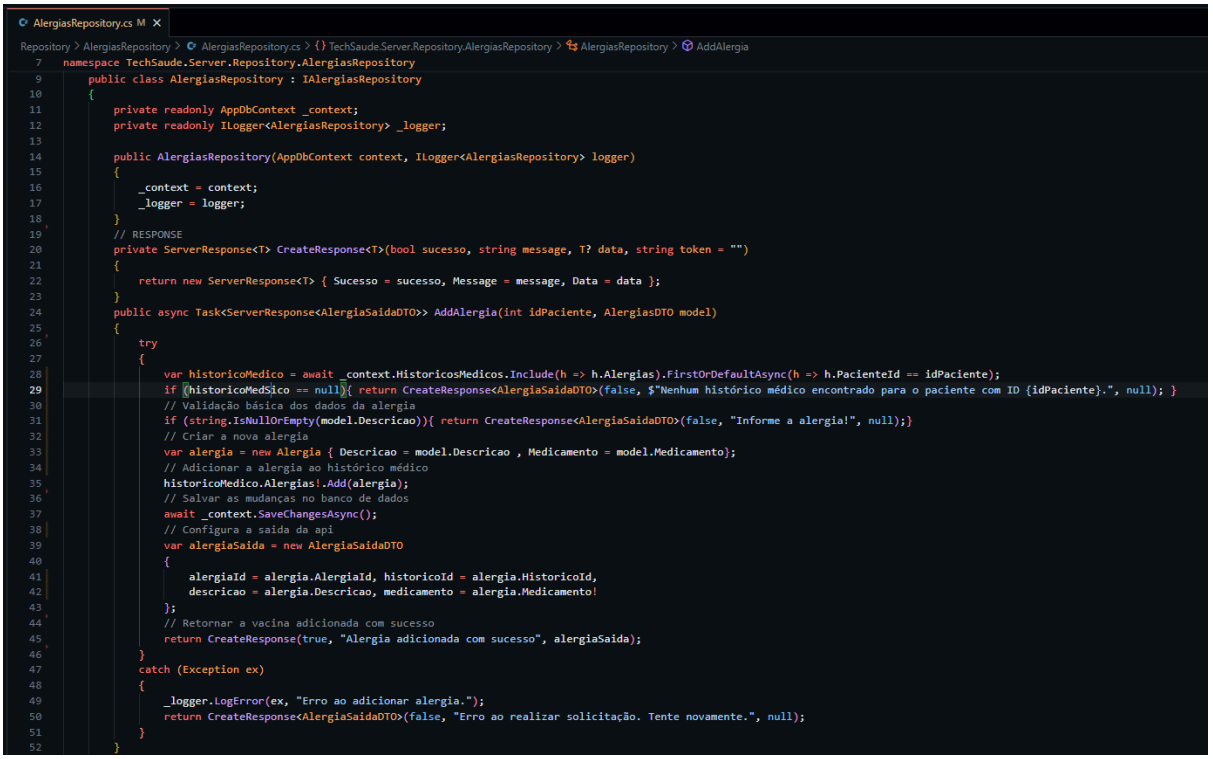
```
using Microsoft.EntityFrameworkCore;
using System.Runtime.InteropServices;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.AlergiasRepository
{
    public class AlergiasRepository : IAlergiasRepository
    {
        private readonly AppDbContext _context;
        private readonly ILogger<AlergiasRepository> _logger;

        public AlergiasRepository(AppDbContext context, ILogger<AlergiasRepository> logger)
        {
            _context = context;
            _logger = logger;
        }
        // RESPONSE
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = "")
        {
            return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };
        }
        public async Task<ServerResponse<AlergiaSaidaDTO>> AddAlergia(int idPaciente, AlergiasDTO model) {···
        }
        public async Task<ServerResponse<List<AlergiaSaidaDTO>>> GetAlergias(int idPaciente){···
        }
        public async Task<ServerResponse<AlergiaSaidaDTO>> UpdateAlergia(int idAlergia, AlergiasDTO model){···
        }
    }
}
```

IAlergiasRepository.cs

Repository > AlergiasRepository > IAlergiasRepository.cs > ...

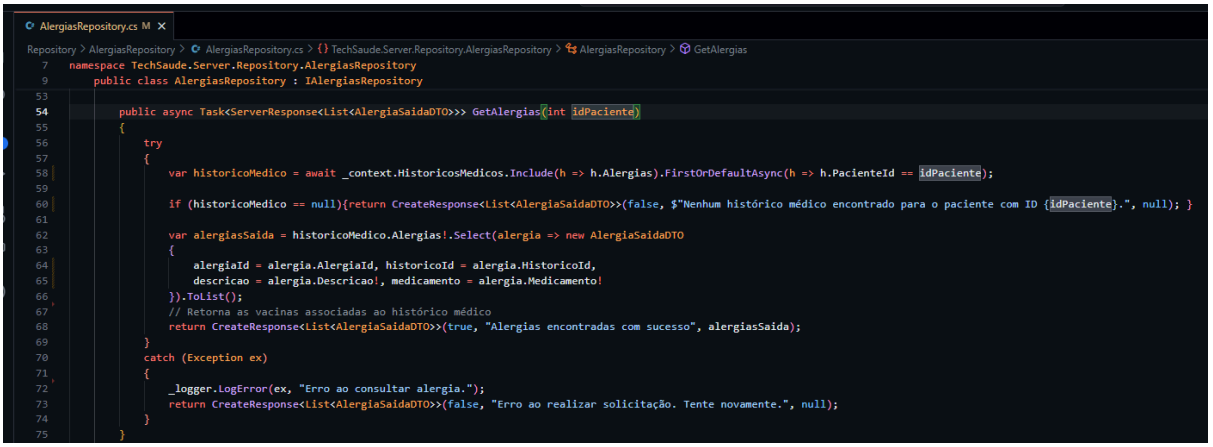
```
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.AlergiasRepository

{
    public interface IAlergiasRepository
    {
        Task<ServerResponse<List<AlergiaSaidaDTO>>> GetAlergias(int historicoId);
        Task<ServerResponse<AlergiaSaidaDTO>> AddAlergia(int id, AlergiasDTO model);
        Task<ServerResponse<AlergiaSaidaDTO>> UpdateAlergia(int id, AlergiasDTO model);
    }
}

```

Anexo 6.1 : Método Adicionar Alergia



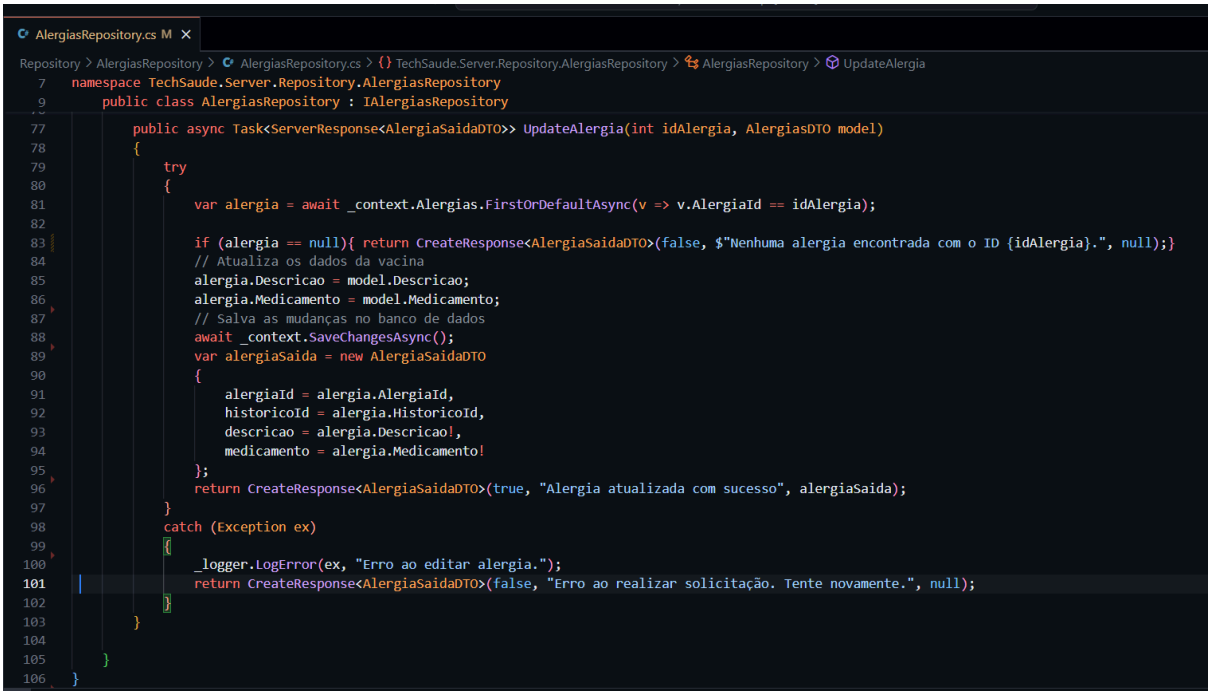
```
AlergiasRepository.cs M
Repository > AlergiasRepository > AlergiasRepository.cs > {} TechSaude.Server.Repository.AlergiasRepository > AlergiasRepository > AddAlergia
namespace TechSaude.Server.Repository.AlergiasRepository
    public class AlergiasRepository : IAlergiasRepository
    {
        private readonly AppDbContext _context;
        private readonly ILogger<AlergiasRepository> _logger;

        public AlergiasRepository(AppDbContext context, ILogger<AlergiasRepository> logger)
        {
            _context = context;
            _logger = logger;
        }
        // RESPONSE
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = "")
        {
            return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };
        }
        public async Task<ServerResponse<AlergiaSaidaDTO>> AddAlergia(int idPaciente, AlergiasDTO model)
        {
            try
            {
                var historicoMedico = await _context.HistoricosMedicos.Include(h => h.Alergias).FirstOrDefaultAsync(h => h.PacienteId == idPaciente);
                if (historicoMedSico == null){ return CreateResponse<AlergiaSaidaDTO>(false, $"Nenhum histórico médico encontrado para o paciente com ID {idPaciente}.", null); }
                // Validação básica dos dados da alergia
                if (string.IsNullOrEmpty(model.Descricao)){ return CreateResponse<AlergiaSaidaDTO>(false, "Informe a alergia!", null);}
                // Criar a nova alergia
                var alergia = new Alergia { Descricao = model.Descricao , Medicamento = model.Medicamento};
                // Adicionar a alergia ao histórico médico
                historicoMedico.Alergias!.Add(alergia);
                // Salvar as mudanças no banco de dados
                await _context.SaveChangesAsync();
                // Configura a saida da api
                var alergiaSaida = new AlergiaSaidaDTO
                {
                    alergiaId = alergia.AlergiaId, historicoId = alergia.HistoricoId,
                    descricao = alergia.Descricao, medicamento = alergia.Medicamento!
                };
                // Retornar a vacina adicionada com sucesso
                return CreateResponse(true, "Alergia adicionada com sucesso", alergiaSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao adicionar alergia.");
                return CreateResponse<AlergiaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 6.2 : Método Consultar Alergia



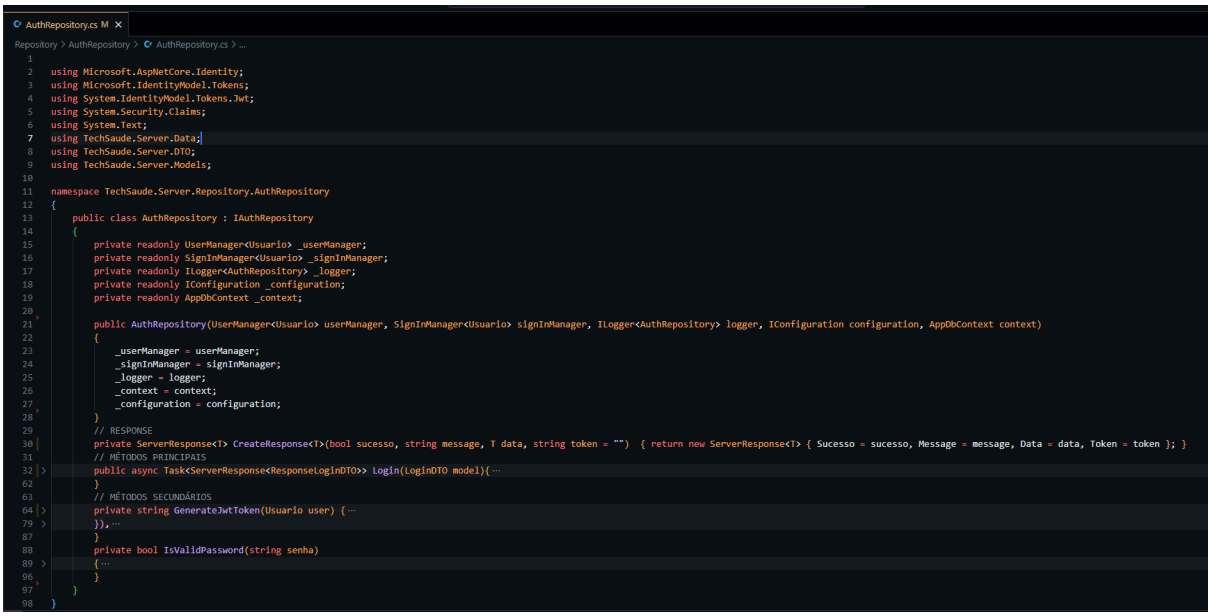
```
AlergiasRepository.cs M
Repository > AlergiasRepository > AlergiasRepository.cs > {} TechSaude.Server.Repository.AlergiasRepository > AlergiasRepository > GetAlergias
namespace TechSaude.Server.Repository.AlergiasRepository
    public class AlergiasRepository : IAlergiasRepository

        public async Task<ServerResponse<List<AlergiaSaidaDTO>>> GetAlergias(int idPaciente)
        {
            try
            {
                var historicoMedico = await _context.HistoricosMedicos.Include(h => h.Alergias).FirstOrDefaultAsync(h => h.PacienteId == idPaciente);

                if (historicoMedico == null){return CreateResponse<List<AlergiaSaidaDTO>>(false, $"Nenhum histórico médico encontrado para o paciente com ID {idPaciente}.", null); }

                var alergiasSaida = historicoMedico.Alergias!.Select(alergia => new AlergiaSaidaDTO
                {
                    alergiaId = alergia.AlergiaId, historicoId = alergia.HistoricoId,
                    descricao = alergia.Descricao!, medicamento = alergia.Medicamento!
                }).ToList();
                // Retorna as vacinas associadas ao histórico médico
                return CreateResponse<List<AlergiaSaidaDTO>>(true, "Alergias encontradas com sucesso", alergiasSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao consultar alergia.");
                return CreateResponse<List<AlergiaSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 6.3 : Método Update Alergia

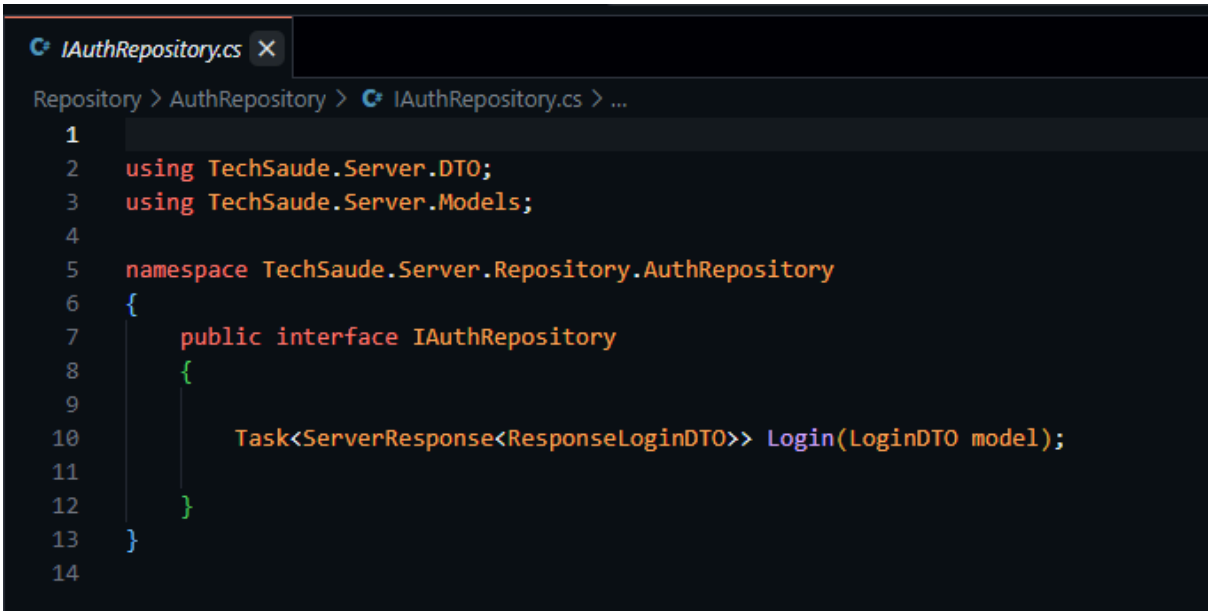


```
AlergiasRepository.cs M
Repository > AlergiasRepository > AlergiasRepository.cs > {} TechSaude.Server.Repository.AlergiasRepository > AlergiasRepository > UpdateAlergia
7   namespace TechSaude.Server.Repository.AlergiasRepository
9       public class AlergiasRepository : IAlergiasRepository
77          public async Task<ServerResponse<AlergiaSaidaDTO>> UpdateAlergia(int idAlergia, AlergiasDTO model)
78          {
79              try
80              {
81                  var alergia = await _context.Alergias.FirstOrDefaultAsync(v => v.AlergiaId == idAlergia);
82
83                  if (alergia == null){ return CreateResponse<AlergiaSaidaDTO>(false, $"Nenhuma alergia encontrada com o ID {idAlergia}.", null);}
84                  // Atualiza os dados da vacina
85                  alergia.Descricao = model.Descricao;
86                  alergia.Medicamento = model.Medicamento;
87                  // Salva as mudanças no banco de dados
88                  await _context.SaveChangesAsync();
89                  var alergiaSaida = new AlergiaSaidaDTO
90                  {
91                      alergiaId = alergia.AlergiaId,
92                      historicoId = alergia.HistoricoId,
93                      descricao = alergia.Descricao!,
94                      medicamento = alergia.Medicamento!
95                  };
96                  return CreateResponse<AlergiaSaidaDTO>(true, "Alergia atualizada com sucesso", alergiaSaida);
97              }
98              catch (Exception ex)
99              {
100                 _logger.LogError(ex, "Erro ao editar alergia.");
101                 return CreateResponse<AlergiaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
102             }
103         }
104
105     }
106 }
```

Anexo 7 - BackEnd : Código do Repositório Auth

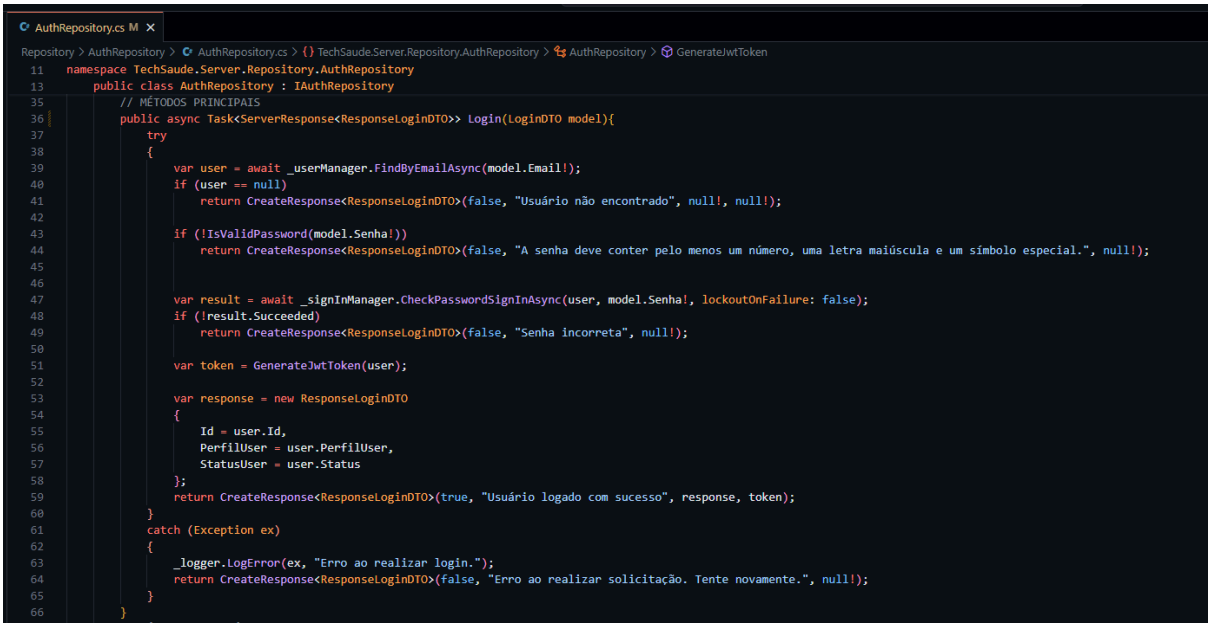


```
AuthRepository.cs M
Repository > AuthRepository > AuthRepository.cs > ...
1
2   using Microsoft.AspNetCore.Identity;
3   using Microsoft.IdentityModel.Tokens;
4   using System.IdentityModel.Tokens.Jwt;
5   using System.Security.Claims;
6   using System.Text;
7   using TechSaude.Server.Data;
8   using TechSaude.Server.DTO;
9   using TechSaude.Server.Models;
10
11  namespace TechSaude.Server.Repository.AuthRepository
12  {
13      public class AuthRepository : IAuthRepository
14      {
15          private readonly UserManager<Usuario> _userManager;
16          private readonly SignInManager<Usuario> _signInManager;
17          private readonly ILogger<AuthRepository> _logger;
18          private readonly IConfiguration _configuration;
19          private readonly AppDbContext _context;
20
21          public AuthRepository(UserManager<Usuario> userManager, SignInManager<Usuario> signInManager, ILogger<AuthRepository> logger, IConfiguration configuration, AppDbContext context)
22          {
23              _userManager = userManager;
24              _signInManager = signInManager;
25              _logger = logger;
26              _context = context;
27              _configuration = configuration;
28          }
29          // RESPONSE
30          private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T data, string token = "")  { return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data, Token = token }; }
31          // MÉTODOS PRINCIPAIS
32          public async Task<ServerResponse<ResponseLoginDTO>> Login(LoginDTO model){…
62          }
63          // MÉTODOS SECUNDÁRIOS
64          private string GenerateJwtToken(Usuario user) {…
79          },…
87          }
88          private bool IsValidPassword(string senha)
89          {…
96          }
97      }
98  }
```



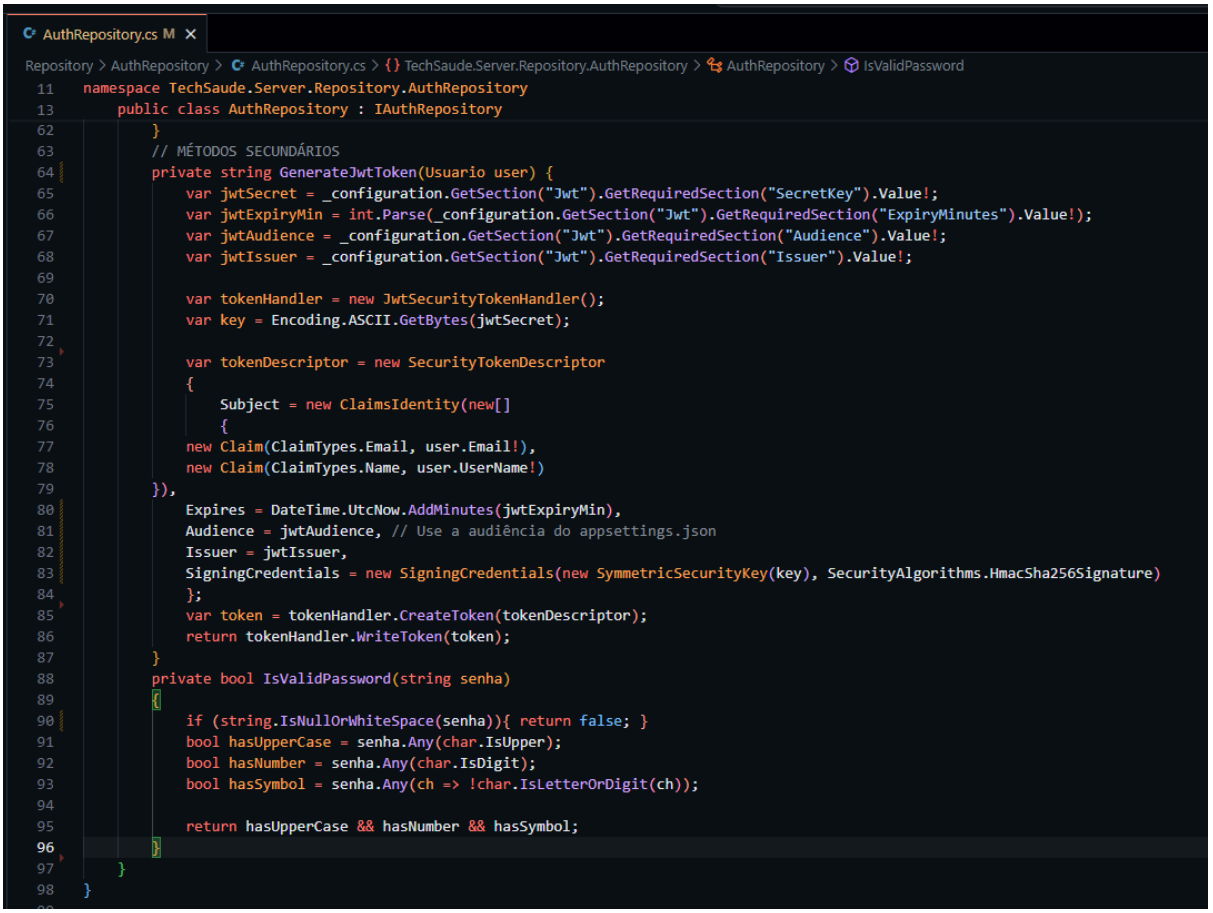
```
IAuthRepository.cs
Repository > AuthRepository > IAuthRepository.cs > ...
1
2   using TechSaude.Server.DTO;
3   using TechSaude.Server.Models;
4
5   namespace TechSaude.Server.Repository.AuthRepository
6   {
7       public interface IAuthRepository
8       {
9
10          Task<ServerResponse<ResponseLoginDTO>> Login(LoginDTO model);
11
12      }
13  }
14
```

Anexo 7.1 : Código do Método Principal (Login)

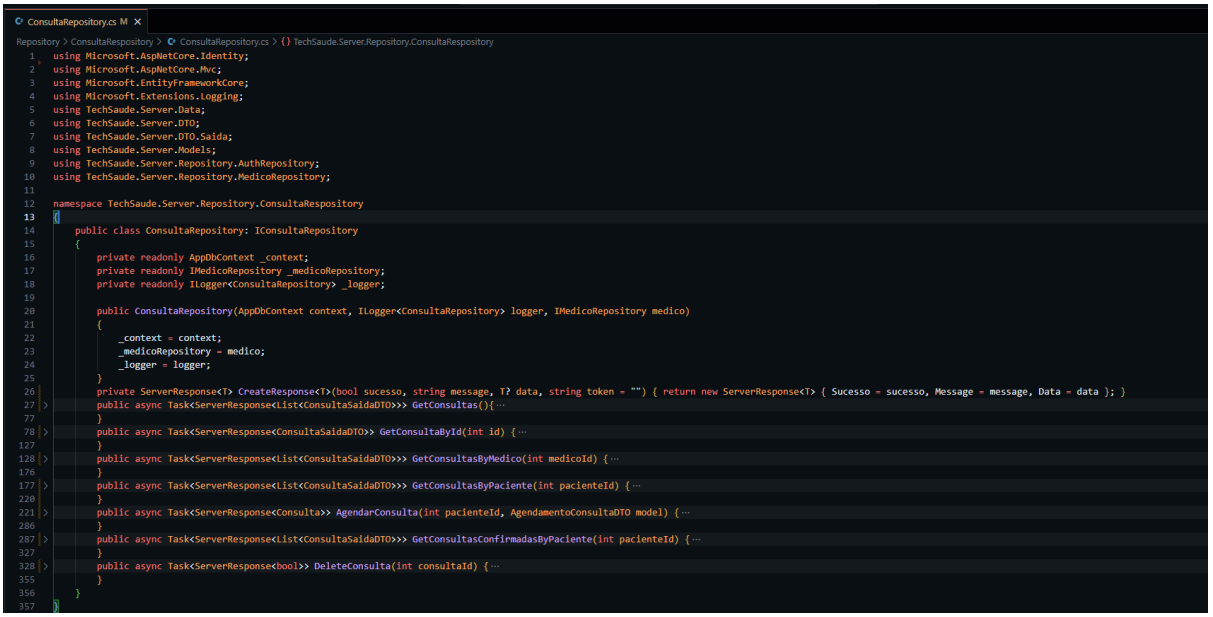


```
AuthRepository.cs M
Repository > AuthRepository > AuthRepository.cs > TechSaude.Server.Repository.AuthRepository > AuthRepository > GenerateJwtToken
11  namespace TechSaude.Server.Repository.AuthRepository
13      public class AuthRepository : IAuthRepository
35          // MÉTODOS PRINCIPAIS
36          public async Task<ServerResponse<ResponseLoginDTO>> Login(LoginDTO model){
37              try
38              {
39                  var user = await _userManager.FindByEmailAsync(model.Email!);
40                  if (user == null)
41                      return CreateResponse<ResponseLoginDTO>(false, "Usuário não encontrado", null!, null!);
42
43                  if (!IsValidPassword(model.Senha!))
44                      return CreateResponse<ResponseLoginDTO>(false, "A senha deve conter pelo menos um número, uma letra maiúscula e um símbolo especial.", null!);
45
46
47                  var result = await _signInManager.CheckPasswordSignInAsync(user, model.Senha!, lockoutOnFailure: false);
48                  if (!result.Succeeded)
49                      return CreateResponse<ResponseLoginDTO>(false, "Senha incorreta", null!);
50
51                  var token = GenerateJwtToken(user);
52
53                  var response = new ResponseLoginDTO
54                  {
55                      Id = user.Id,
56                      PerfilUser = user.PerfilUser,
57                      StatusUser = user.Status
58                  };
59                  return CreateResponse<ResponseLoginDTO>(true, "Usuário logado com sucesso", response, token);
60              }
61              catch (Exception ex)
62              {
63                  _logger.LogError(ex, "Erro ao realizar login.");
64                  return CreateResponse<ResponseLoginDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null!);
65              }
66          }
```

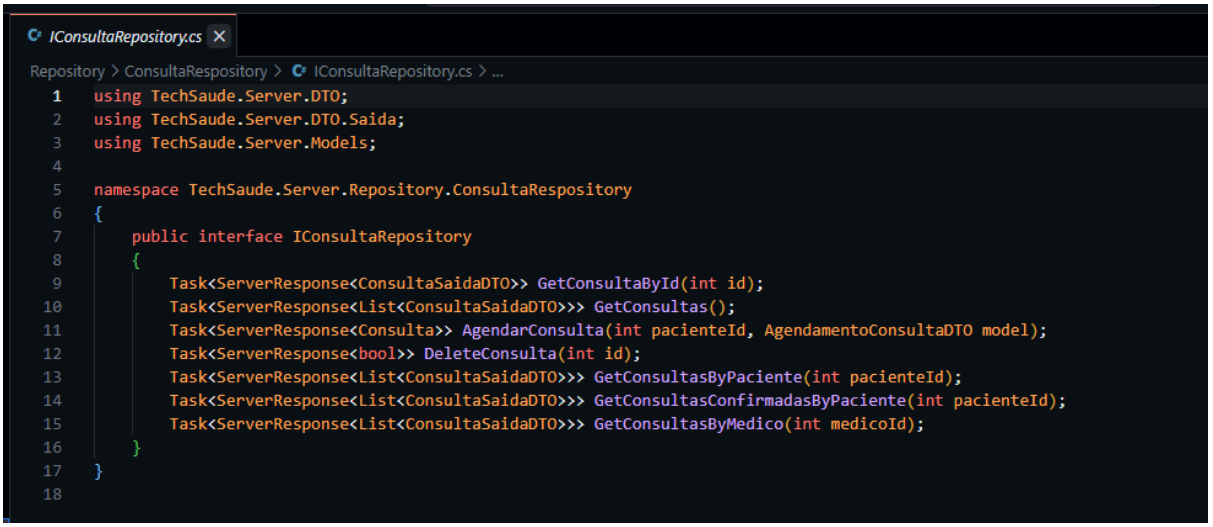
Anexo 7.1.1 : Código dos Métodos Secundários



Anexo 8 - BackEnd : Código do Repositório Consultas



```
ConsultaRepository.cs M
Repository > ConsultaRespository > ConsultaRepository.cs > {} TechSaude.Server.Repository.ConsultaRespository
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.AuthRepository;
using TechSaude.Server.Repository.MedicoRepository;

namespace TechSaude.Server.Repository.ConsultaRespository
{
    public class ConsultaRepository: IConsultaRepository
    {
        private readonly AppDbContext _context;
        private readonly IMedicoRepository _medicoRepository;
        private readonly ILogger<ConsultaRepository> _logger;

        public ConsultaRepository(AppDbContext context, ILogger<ConsultaRepository> logger, IMedicoRepository medico)
        {
            _context = context;
            _medicoRepository = medico;
            _logger = logger;
        }
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = "") { return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data }; }
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultas(){...
        }
        public async Task<ServerResponse<ConsultaSaidaDTO>> GetConsultaById(int id) {...
        }
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasByMedico(int medicoId) {...
        }
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasByPaciente(int pacienteId) {...
        }
        public async Task<ServerResponse<Consulta>> AgendarConsulta(int pacienteId, AgendamentoConsultaDTO model) {...
        }
        public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasConfirmadasByPaciente(int pacienteId) {...
        }
        public async Task<ServerResponse<bool>> DeleteConsulta(int consultaId) {...
        }
    }
}
```



```
IConsultaRepository.cs
Repository > ConsultaRespository > IConsultaRepository.cs > ...
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.ConsultaRespository
{
    public interface IConsultaRepository
    {
        Task<ServerResponse<ConsultaSaidaDTO>> GetConsultaById(int id);
        Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultas();
        Task<ServerResponse<Consulta>> AgendarConsulta(int pacienteId, AgendamentoConsultaDTO model);
        Task<ServerResponse<bool>> DeleteConsulta(int id);
        Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasByPaciente(int pacienteId);
        Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasConfirmadasByPaciente(int pacienteId);
        Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasByMedico(int medicoId);
    }
}
```

Anexo 8.1 : GetConsultas
Anexo 8.2 : GetConsultasByID

ConsultaRepository.cs M

Repository > ConsultaRespository > ConsultaRepository.cs > {} TechSaude.Server.Repository.ConsultaRespository > ConsultaRepository > GetConsultaById

```
12  namespace TechSaude.Server.Repository.ConsultaRespository
14      public class ConsultaRepository: IConsultaRepository
64          public async Task<ServerResponse<ConsultaSaidaDTO>> GetConsultaById(int id) {
66              {
67                  var consulta = await _context.Consultas.Include(c => c.Paciente).Include(c => c.Medico).FirstOrDefaultAsync(c => c.Id == id);
68
69                  if (consulta == null){ return CreateResponse<ConsultaSaidaDTO>(false, "Consulta não encontrada", null); }
70
71                  var consultaDTO = new ConsultaSaidaDTO
72                  {
73                      Id = consulta.Id,
74                      PacienteId = consulta.PacienteId,
75                      MedicoId = consulta.MedicoId,
76                      DataHora = consulta.DataHora,
77                      Status = consulta.Status,
78                      Local = consulta.Local,
79                      Encaminhamentos = consulta.Encaminhamentos!,
80                      Paciente = new PacienteDTO {
81                          Id = consulta.Paciente!.Id,
82                          NomeCompleto = consulta.Paciente.NomeCompleto,
83                          DataNascimento = consulta.Paciente.DataNascimento,
84                          Sexo = consulta.Paciente.Sexo,
85                          Endereco = consulta.Paciente.Endereco,
86                          Telefone = consulta.Paciente.Telefone,
87                          Cns = consulta.Paciente.CNS,
88                          Convenio = consulta.Paciente.Convenio
89                      },
90                      Medico = new MedicoDTO {
91                          Id = consulta.Medico!.Id,
92                          Nome = consulta.Medico.NomeCompleto,
93                          Especialidade = consulta.Medico.Especialidade
94                      }
95                  };
96                  return CreateResponse<ConsultaSaidaDTO>(true, "Consulta encontrada com sucesso.", consultaDTO);
97              }
98              catch (Exception ex)
99              {
100                 _logger.LogError(ex, "Erro ao pegar consultas.");
101                 return CreateResponse<ConsultaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null!);
102             }
103         }
```

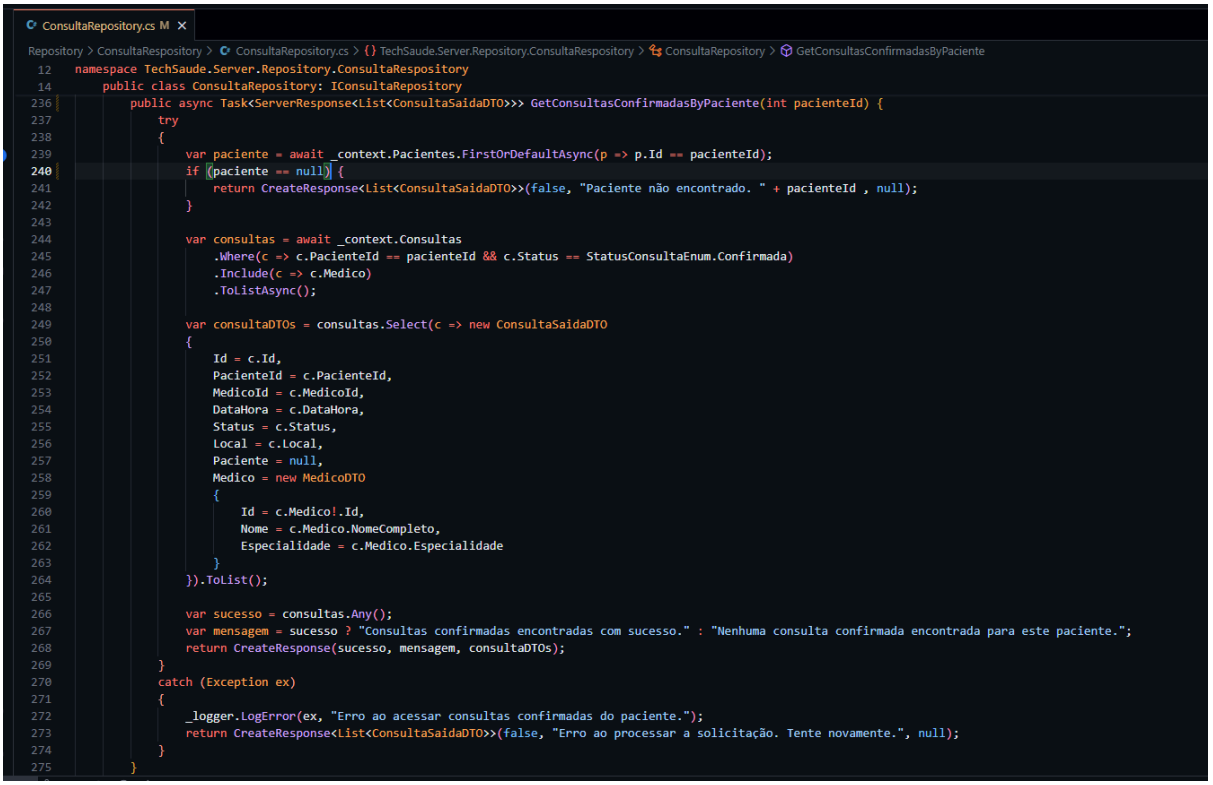
Anexo 8.3 : AgendarConsulta

ConsultaRepository.cs M

Repository > ConsultaRespository > ConsultaRepository.cs > {} TechSaude.Server.Repository.ConsultaRespository > ConsultaRepository > AgendarConsulta

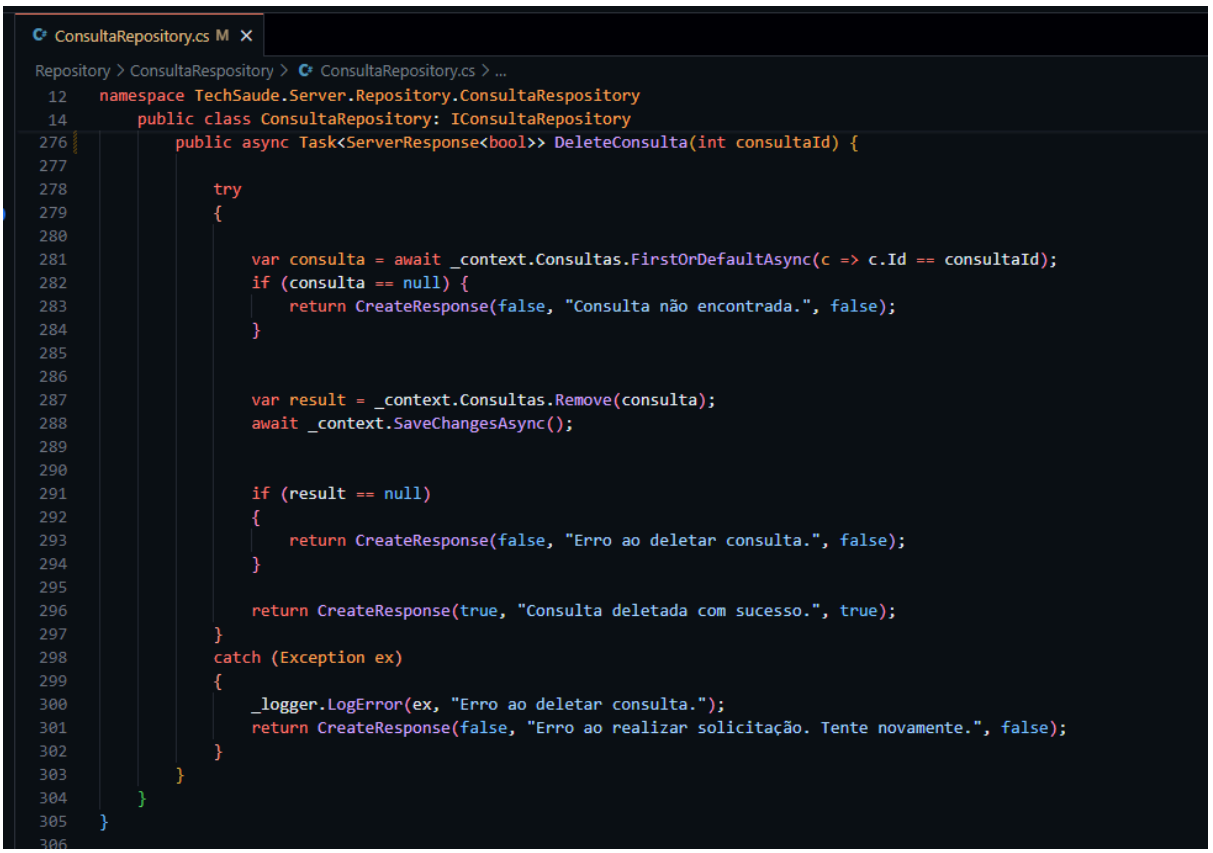
```
12  namespace TechSaude.Server.Repository.ConsultaRespository
14      public class ConsultaRepository: IConsultaRepository
197         public async Task<ServerResponse<Consulta>> AgendarConsulta(int pacienteId, AgendamentoConsultaDTO model) {
198             try{
199                 // Verificar data
200                 if (model.DataHora <= DateTime.Now){ return CreateResponse<Consulta>(false, "A data e hora da consulta devem ser no futuro.", null);}
201                 // Verificar se o médico existe
202                 var medico = await _context.Medicos.FirstOrDefaultAsync(m => m.Id == model.MedicoId);
203                 if (medico == null)  return CreateResponse<Consulta>(false, $"Médico com ID {model.MedicoId} não encontrado.", null);
204                 // Verificar se o paciente existe
205                 var paciente = await _context.Pacientes.FirstOrDefaultAsync(p => p.Id == model.PacienteId);
206                 if (paciente == null) return CreateResponse<Consulta>(false, $"Paciente com ID {model.PacienteId} não encontrado.", null);
207                 // Criar uma nova consulta
208                 var consulta = new Consulta {
209                     DataHora = model.DataHora,PacienteId = model.PacienteId, MedicoId = model.MedicoId,Local = model.Local, Status = StatusConsultaEnum.Agendada,
210                 };
211                 // Adicionar a consulta ao contexto
212                 _context.Consultas.Add(consulta);
213                 await _context.SaveChangesAsync();
214                 var message = "Consulta agendada com sucesso!";
215                 var sucesso = true;
216                 // Tentar confirmar a consulta após o agendamento
217                 try {
218                     var confirmacaoResponse = await _medicoRepository.ConfirmarConsulta(consulta.Id);
219                     if (!confirmacaoResponse.Sucesso){ sucesso = false;
220                         message += " No entanto, houve um problema ao confirmar a consulta: " + confirmacaoResponse.Message;
221                     }
222                     else{message += "Além disso, " + confirmacaoResponse.Message; }
223                 }
224                 catch (Exception ex){
225                     _logger.LogError(ex, "Erro ao confirmar consulta.");
226                     message += " No entanto, ocorreu um erro ao confirmar a consulta. Tente novamente ";
227                     sucesso = false;
228                 }
229                 return CreateResponse<Consulta>(sucesso, message, consulta);
230             }
231             catch (Exception ex) {
232                 _logger.LogError(ex, "Erro ao agendar consulta.");
233                 return CreateResponse<Consulta>(false, "Erro ao processar a solicitação. Tente novamente.", null);
234             }
235         }
```

Anexo 8.4 : Get Consultas Confirmadas By Paciente



```
ConsultaRepository.cs M
Repository > ConsultaRespository > ConsultaRepository.cs > { } TechSaude.Server.Repository.ConsultaRespository > ConsultaRepository > GetConsultasConfirmadasByPaciente
12  namespace TechSaude.Server.Repository.ConsultaRespository
14      public class ConsultaRepository: IConsultaRepository
236         public async Task<ServerResponse<List<ConsultaSaidaDTO>>> GetConsultasConfirmadasByPaciente(int pacienteId) {
237             try
238             {
239                 var paciente = await _context.Pacientes.FirstOrDefaultAsync(p => p.Id == pacienteId);
240                 if (paciente == null) {
241                     return CreateResponse<List<ConsultaSaidaDTO>>(false, "Paciente não encontrado. " + pacienteId , null);
242                 }
243
244                 var consultas = await _context.Consultas
245                     .Where(c => c.PacienteId == pacienteId && c.Status == StatusConsultaEnum.Confirmada)
246                     .Include(c => c.Medico)
247                     .ToListAsync();
248
249                 var consultaDTOs = consultas.Select(c => new ConsultaSaidaDTO
250                 {
251                     Id = c.Id,
252                     PacienteId = c.PacienteId,
253                     MedicoId = c.MedicoId,
254                     DataHora = c.DataHora,
255                     Status = c.Status,
256                     Local = c.Local,
257                     Paciente = null,
258                     Medico = new MedicoDTO
259                     {
260                         Id = c.Medico!.Id,
261                         Nome = c.Medico.NomeCompleto,
262                         Especialidade = c.Medico.Especialidade
263                     }
264                 }).ToList();
265
266                 var sucesso = consultas.Any();
267                 var mensagem = sucesso ? "Consultas confirmadas encontradas com sucesso." : "Nenhuma consulta confirmada encontrada para este paciente.";
268                 return CreateResponse(sucesso, mensagem, consultaDTOs);
269             }
270             catch (Exception ex)
271             {
272                 _logger.LogError(ex, "Erro ao acessar consultas confirmadas do paciente.");
273                 return CreateResponse<List<ConsultaSaidaDTO>>(false, "Erro ao processar a solicitação. Tente novamente.", null);
274             }
275         }
```

Anexo 8.5 : Delete Consulta



```
ConsultaRepository.cs M
Repository > ConsultaRespository > ConsultaRepository.cs > ...
12  namespace TechSaude.Server.Repository.ConsultaRespository
14      public class ConsultaRepository: IConsultaRepository
276         public async Task<ServerResponse<bool>> DeleteConsulta(int consultaId) {
277
278             try
279             {
280
281                 var consulta = await _context.Consultas.FirstOrDefaultAsync(c => c.Id == consultaId);
282                 if (consulta == null) {
283                     return CreateResponse(false, "Consulta não encontrada.", false);
284                 }
285
286
287                 var result = _context.Consultas.Remove(consulta);
288                 await _context.SaveChangesAsync();
289
290
291                 if (result == null)
292                 {
293                     return CreateResponse(false, "Erro ao deletar consulta.", false);
294                 }
295
296                 return CreateResponse(true, "Consulta deletada com sucesso.", true);
297             }
298             catch (Exception ex)
299             {
300                 _logger.LogError(ex, "Erro ao deletar consulta.");
301                 return CreateResponse(false, "Erro ao realizar solicitação. Tente novamente.", false);
302             }
303         }
304     }
305 }
306
```

Anexo 9 - BackEnd : **Repositório de Documentos**

DocumentoRepository.cs M

Repository > DocumentosRepository > DocumentoRepository.cs > ...

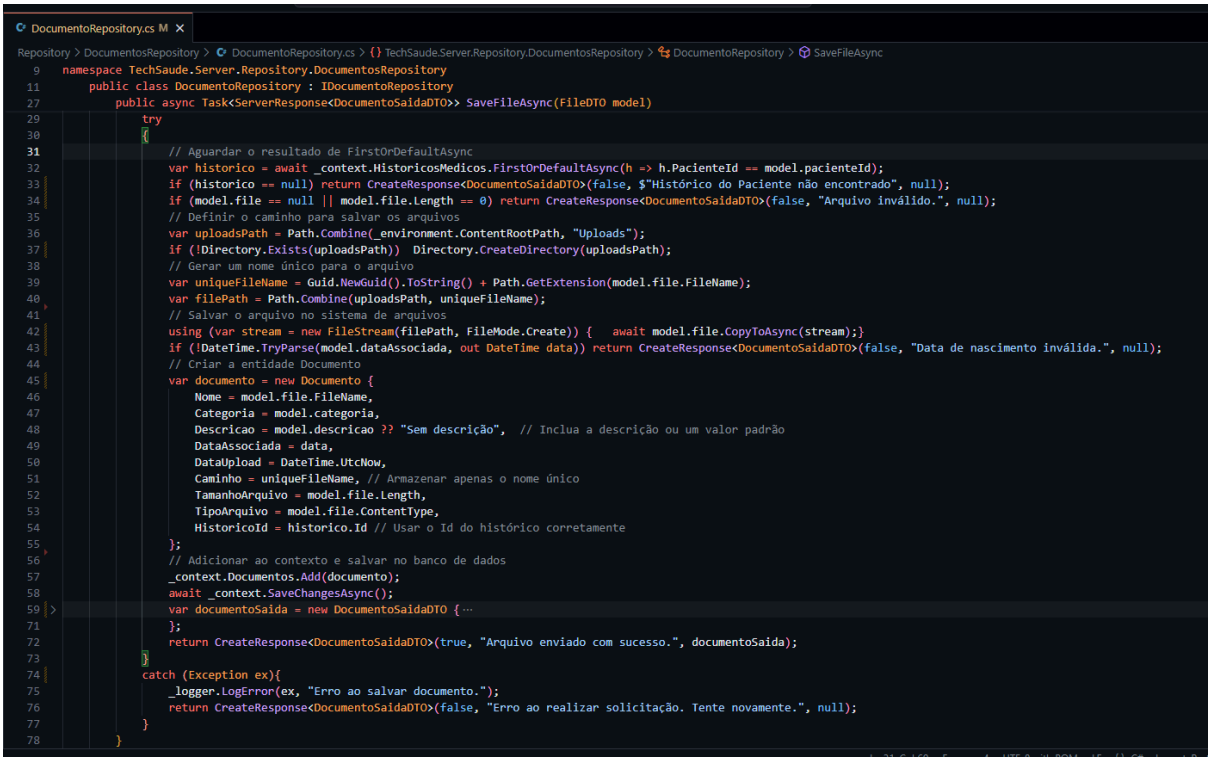
```
using Microsoft.EntityFrameworkCore;
using Mysqlx.Expr;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using static Azure.Core.HttpHeader;

namespace TechSaude.Server.Repository.DocumentosRepository
{
    public class DocumentoRepository : IDocumentoRepository
    {
        private readonly IWebHostEnvironment _environment;
        private readonly AppDbContext _context;
        private readonly ILogger<DocumentoRepository> _logger;

        public DocumentoRepository(IWebHostEnvironment environment, AppDbContext context, ILogger<DocumentoRepository> logger)
        {
            _environment = environment;
            _context = context;
            _logger = logger;
        }
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = "")
        {
            return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };
        }
        public async Task<ServerResponse<DocumentoSaidaDTO>> SaveFileAsync(FileDTO model)
        {...
        }
        public async Task<ServerResponse<DocumentoSaidaDTO>> GetDocumento(int id){...
        }
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetExames(int pacienteId)
        {...
        }
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetReceitas(int pacienteId){...
        }
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetOutros(int pacienteId)
        {...
        }
    }
}
```

IDocumentoRepository.cs M

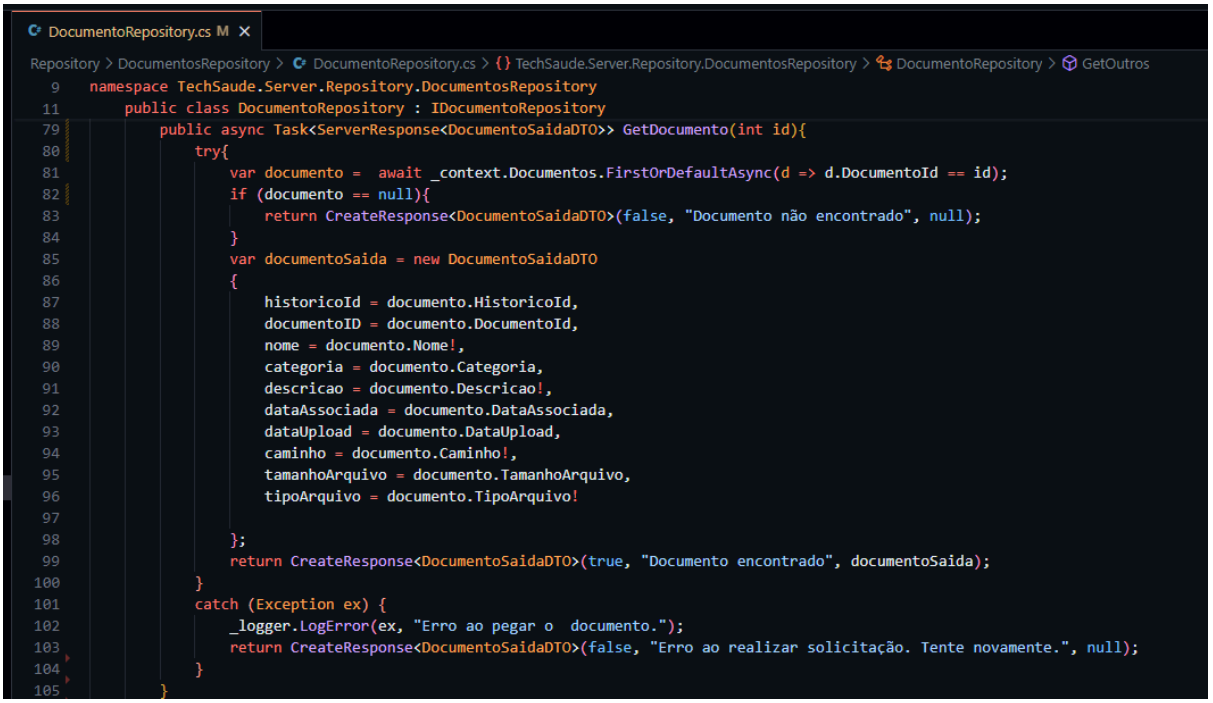
Repository > DocumentosRepository > IDocumentoRepository.cs > { } TechSaude.Server.Repository.DocumentosRepository > IDocumentoRepository > GetOutros

```
using Microsoft.AspNetCore.Mvc;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.DocumentosRepository
{
    public interface IDocumentoRepository
    {
        Task<ServerResponse<DocumentoSaidaDTO>> SaveFileAsync(FileDTO model);
        Task<ServerResponse<DocumentoSaidaDTO>> GetDocumento(int id);
        Task<ServerResponse<List<DocumentoSaidaDTO>>> GetExames(int pacienteId);
        Task<ServerResponse<List<DocumentoSaidaDTO>>> GetReceitas(int pacienteId);
        Task<ServerResponse<List<DocumentoSaidaDTO>>> GetOutros(int pacienteId);
    }
}
```

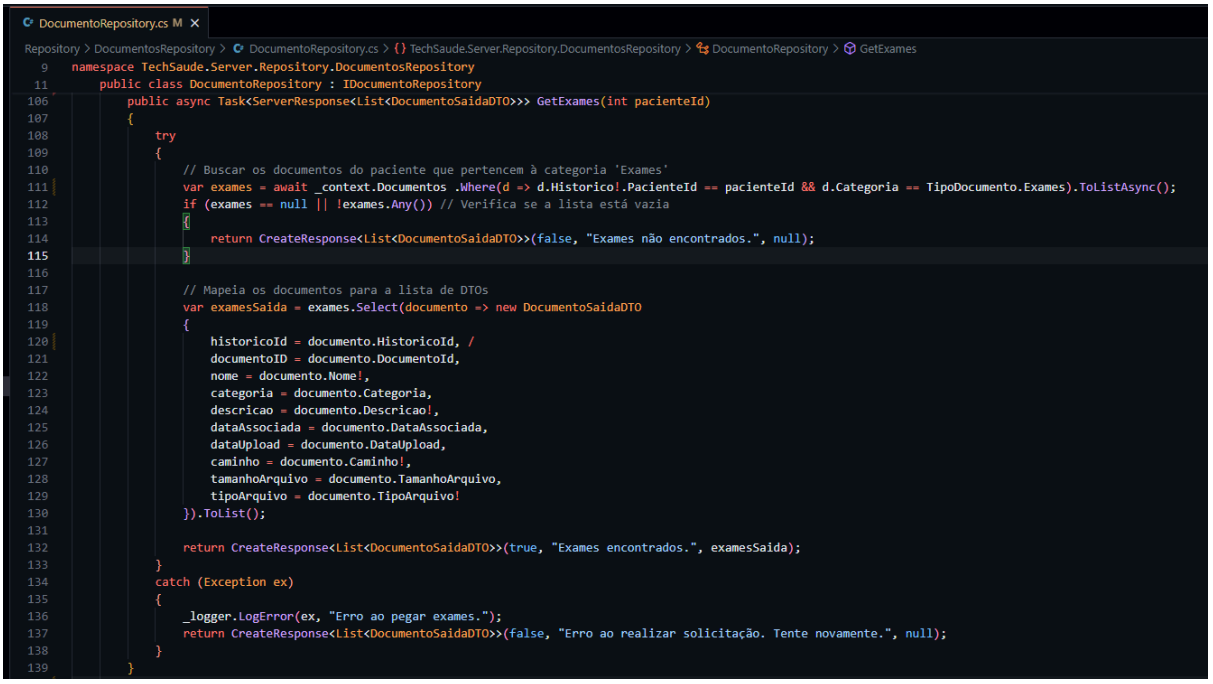
Anexo 9.1 : SaveAsync



Anexo 9.2 : Get Documento

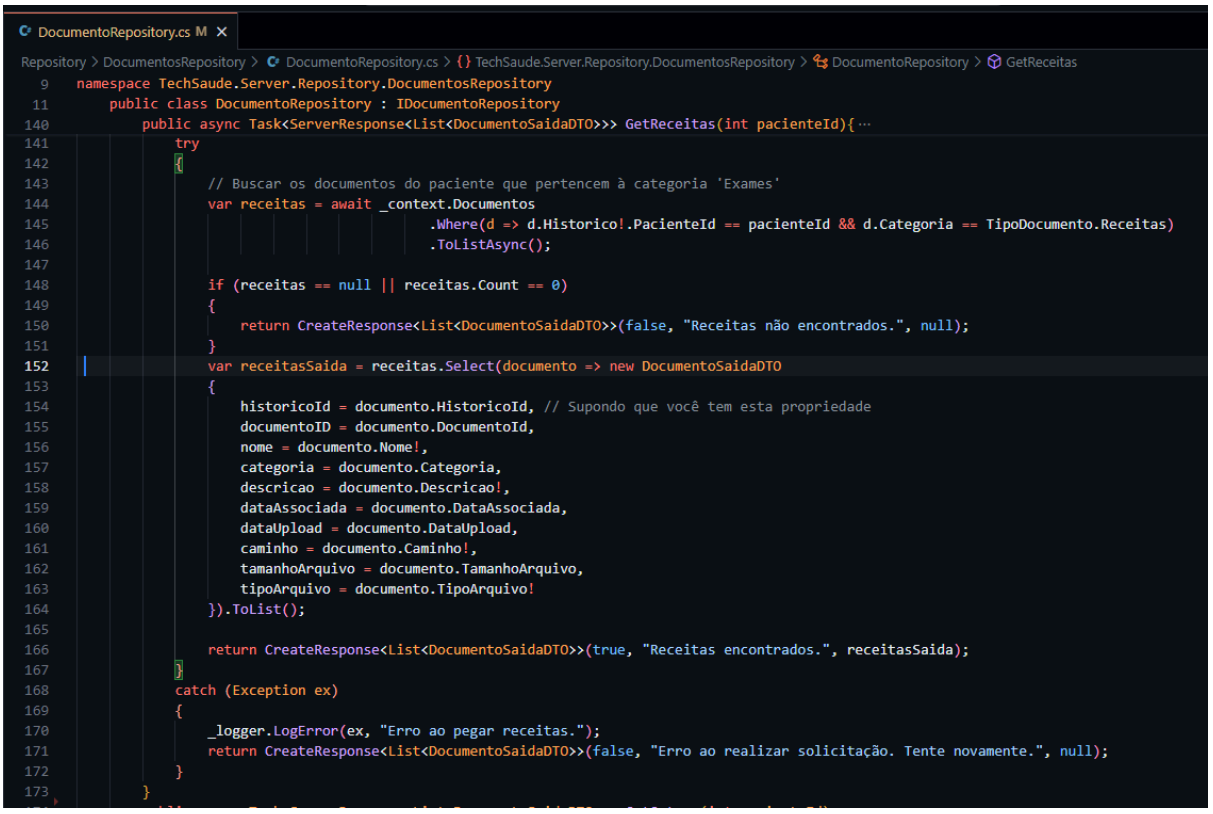


Anexo 9.3 : Get Exames



```
DocumentoRepository.cs M
Repository > DocumentosRepository > DocumentoRepository.cs > {} TechSaude.Server.Repository.DocumentosRepository > DocumentoRepository > GetExames
namespace TechSaude.Server.Repository.DocumentosRepository
    public class DocumentoRepository : IDocumentoRepository
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetExames(int pacienteId)
        {
            try
            {
                // Buscar os documentos do paciente que pertencem à categoria 'Exames'
                var exames = await _context.Documentos .Where(d => d.Historico!.PacienteId == pacienteId && d.Categoria == TipoDocumento.Exames).ToListAsync();
                if (exames == null || !exames.Any()) // Verifica se a lista está vazia
                {
                    return CreateResponse<List<DocumentoSaidaDTO>>(false, "Exames não encontrados.", null);
                }

                // Mapeia os documentos para a lista de DTOs
                var examesSaida = exames.Select(documento => new DocumentoSaidaDTO
                {
                    historicoId = documento.HistoricoId, /
                    documentoID = documento.DocumentoId,
                    nome = documento.Nome!,
                    categoria = documento.Categoria,
                    descricao = documento.Descricao!,
                    dataAssociada = documento.DataAssociada,
                    dataUpload = documento.DataUpload,
                    caminho = documento.Caminho!,
                    tamanhoArquivo = documento.TamanhoArquivo,
                    tipoArquivo = documento.TipoArquivo!
                }).ToList();

                return CreateResponse<List<DocumentoSaidaDTO>>(true, "Exames encontrados.", examesSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao pegar exames.");
                return CreateResponse<List<DocumentoSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 9.4 : Get Receitas



```
DocumentoRepository.cs M
Repository > DocumentosRepository > DocumentoRepository.cs > {} TechSaude.Server.Repository.DocumentosRepository > DocumentoRepository > GetReceitas
namespace TechSaude.Server.Repository.DocumentosRepository
    public class DocumentoRepository : IDocumentoRepository
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetReceitas(int pacienteId){ ···
            try
            {
                // Buscar os documentos do paciente que pertencem à categoria 'Exames'
                var receitas = await _context.Documentos
                                             .Where(d => d.Historico!.PacienteId == pacienteId && d.Categoria == TipoDocumento.Receitas)
                                             .ToListAsync();

                if (receitas == null || receitas.Count == 0)
                {
                    return CreateResponse<List<DocumentoSaidaDTO>>(false, "Receitas não encontrados.", null);
                }
                var receitasSaida = receitas.Select(documento => new DocumentoSaidaDTO
                {
                    historicoId = documento.HistoricoId, // Supondo que você tem esta propriedade
                    documentoID = documento.DocumentoId,
                    nome = documento.Nome!,
                    categoria = documento.Categoria,
                    descricao = documento.Descricao!,
                    dataAssociada = documento.DataAssociada,
                    dataUpload = documento.DataUpload,
                    caminho = documento.Caminho!,
                    tamanhoArquivo = documento.TamanhoArquivo,
                    tipoArquivo = documento.TipoArquivo!
                }).ToList();

                return CreateResponse<List<DocumentoSaidaDTO>>(true, "Receitas encontrados.", receitasSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao pegar receitas.");
                return CreateResponse<List<DocumentoSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 9.5 : Get Outros

DocumentoRepository.cs M

Repository > DocumentosRepository > DocumentoRepository.cs > {} TechSaude.Server.Repository.DocumentosRepository > DocumentoRepository > GetOutros

```
namespace TechSaude.Server.Repository.DocumentosRepository
    public class DocumentoRepository : IDocumentoRepository
        public async Task<ServerResponse<List<DocumentoSaidaDTO>>> GetOutros(int pacienteId)
        {
            try
            {
                // Buscar os documentos do paciente que pertencem à categoria 'Exames'
                var outros = await _context.Documentos
                                           .Where(d => d.Historico!.PacienteId == pacienteId && d.Categoria == TipoDocumento.Outros)
                                           .ToListAsync();

                if (outros == null || outros.Count == 0)
                {
                    return CreateResponse<List<DocumentoSaidaDTO>>(false, "Documentos não encontrados.", null);
                }

                var dcoumentosSaida = outros.Select(documento => new DocumentoSaidaDTO
                {
                    historicoId = documento.HistoricoId, // Supondo que você tem esta propriedade
                    documentoID = documento.DocumentoId,
                    nome = documento.Nome!,
                    categoria = documento.Categoria,
                    descricao = documento.Descricao!,
                    dataAssociada = documento.DataAssociada,
                    dataUpload = documento.DataUpload,
                    caminho = documento.Caminho!,
                    tamanhoArquivo = documento.TamanhoArquivo,
                    tipoArquivo = documento.TipoArquivo!
                }).ToList();
                return CreateResponse<List<DocumentoSaidaDTO>>(true, "Documentos encontrados.", dcoumentosSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao pegar documentos.");
                return CreateResponse<List<DocumentoSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
    }
}
```

Anexo 10 - BackEnd : **Repositório de Doenças**

DoencasRepository.cs M

Repository > DoencasRepository > DoencasRepository.cs > {} TechSaude.Server.Repository.DoencasRepository > DoencasRepository > CreateResponse

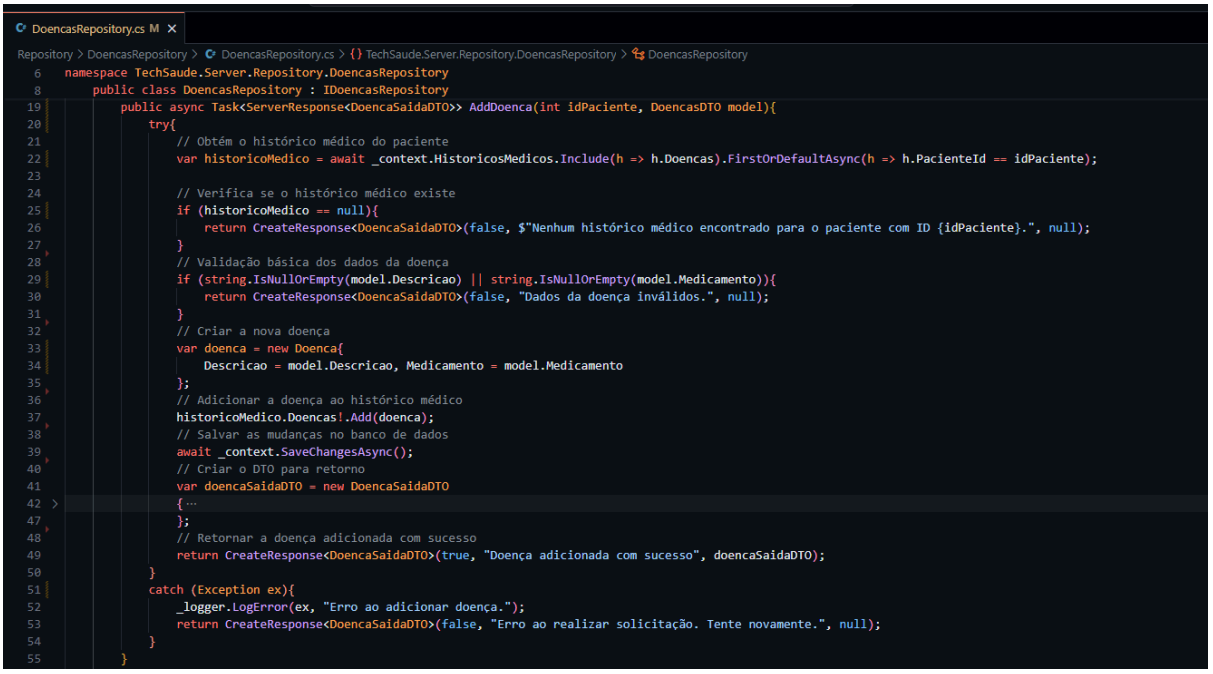
```
using Microsoft.EntityFrameworkCore;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.DoencasRepository
{
    public class DoencasRepository : IDoencasRepository
    {
        private readonly AppDbContext _context;
        private readonly ILogger<DoencasRepository> _logger;

        public DoencasRepository(AppDbContext context, ILogger<DoencasRepository> logger){
            _context = context;
            _logger = logger;
        }
        // RESPONSE
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = ""){
            return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };}
        public async Task<ServerResponse<DoencaSaidaDTO>> AddDoenca(int idPaciente, DoencasDTO model){ ···
        }
        public async Task<ServerResponse<List<DoencaSaidaDTO>>> GetDoencas(int idPaciente)
        { ···
        }
        public async Task<ServerResponse<DoencaSaidaDTO>> UpdateDoenca(int id, DoencasDTO model)
        { ···
        }
    }
}
```

IDoencasRepository.cs

Repository > DoencasRepository > IDoencasRepository.cs > ...

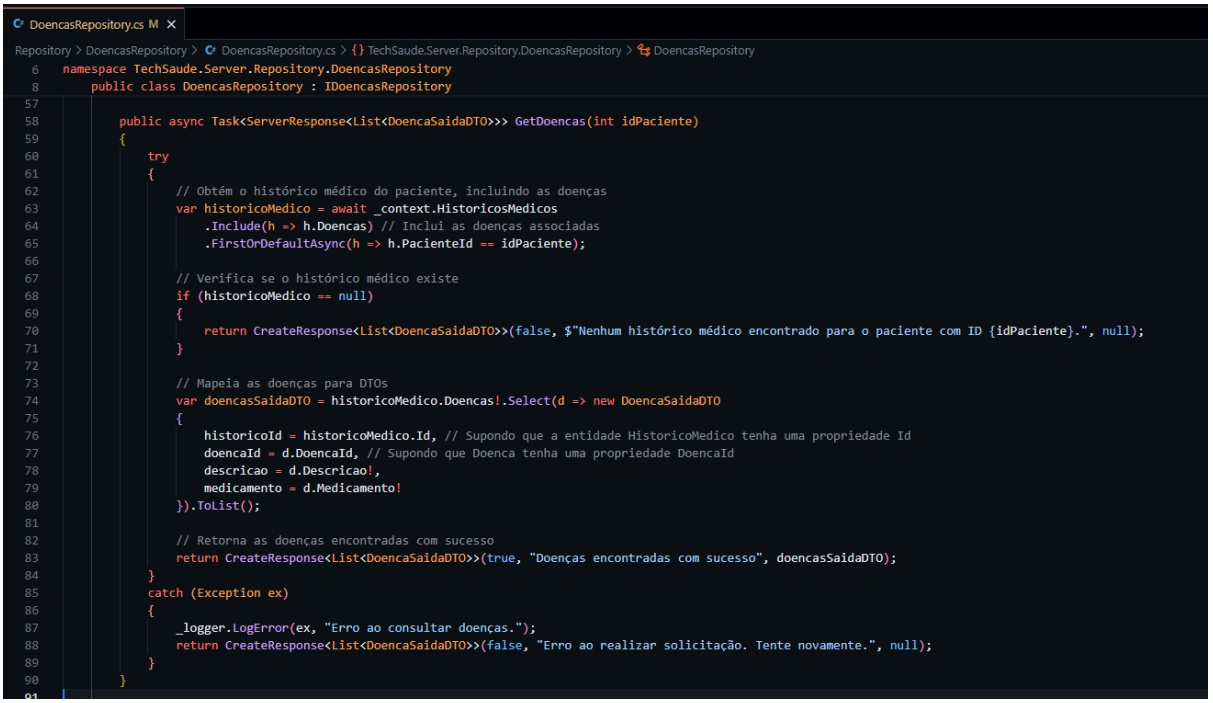
```
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.DoencasRepository
{
    public interface IDoencasRepository
    {
        Task<ServerResponse<List<DoencaSaidaDTO>>> GetDoencas(int historicoId);
        Task<ServerResponse<DoencaSaidaDTO>> AddDoenca(int idPaciente, DoencasDTO model);
        Task<ServerResponse<DoencaSaidaDTO>> UpdateDoenca(int id, DoencasDTO model);
    }
}
```

Anexo 10.1 : Add Doenças



```
DoencasRepository.cs M
Repository > DoencasRepository > DoencasRepository.cs > {} TechSaude.Server.Repository.DoencasRepository > DoencasRepository
namespace TechSaude.Server.Repository.DoencasRepository
    public class DoencasRepository : IDoencasRepository
        public async Task<ServerResponse<DoencaSaidaDTO>> AddDoenca(int idPaciente, DoencasDTO model){
            try{
                // Obtém o histórico médico do paciente
                var historicoMedico = await _context.HistoricosMedicos.Include(h => h.Doencas).FirstOrDefaultAsync(h => h.PacienteId == idPaciente);

                // Verifica se o histórico médico existe
                if (historicoMedico == null){
                    return CreateResponse<DoencaSaidaDTO>(false, $"Nenhum histórico médico encontrado para o paciente com ID {idPaciente}.", null);
                }
                // Validação básica dos dados da doença
                if (string.IsNullOrEmpty(model.Descricao) || string.IsNullOrEmpty(model.Medicamento)){
                    return CreateResponse<DoencaSaidaDTO>(false, "Dados da doença inválidos.", null);
                }
                // Criar a nova doença
                var doenca = new Doenca{
                    Descricao = model.Descricao, Medicamento = model.Medicamento
                };
                // Adicionar a doença ao histórico médico
                historicoMedico.Doencas!.Add(doenca);
                // Salvar as mudanças no banco de dados
                await _context.SaveChangesAsync();
                // Criar o DTO para retorno
                var doencaSaidaDTO = new DoencaSaidaDTO
                {...
                };
                // Retornar a doença adicionada com sucesso
                return CreateResponse<DoencaSaidaDTO>(true, "Doença adicionada com sucesso", doencaSaidaDTO);
            }
            catch (Exception ex){
                _logger.LogError(ex, "Erro ao adicionar doença.");
                return CreateResponse<DoencaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 10.2 : GetDoencas



```
DoencasRepository.cs M
Repository > DoencasRepository > DoencasRepository.cs > {} TechSaude.Server.Repository.DoencasRepository > DoencasRepository
namespace TechSaude.Server.Repository.DoencasRepository
    public class DoencasRepository : IDoencasRepository

        public async Task<ServerResponse<List<DoencaSaidaDTO>>> GetDoencas(int idPaciente)
        {
            try
            {
                // Obtém o histórico médico do paciente, incluindo as doenças
                var historicoMedico = await _context.HistoricosMedicos
                    .Include(h => h.Doencas) // Inclui as doenças associadas
                    .FirstOrDefaultAsync(h => h.PacienteId == idPaciente);

                // Verifica se o histórico médico existe
                if (historicoMedico == null)
                {
                    return CreateResponse<List<DoencaSaidaDTO>>(false, $"Nenhum histórico médico encontrado para o paciente com ID {idPaciente}.", null);
                }

                // Mapeia as doenças para DTOs
                var doencasSaidaDTO = historicoMedico.Doencas!.Select(d => new DoencaSaidaDTO
                {
                    historicoId = historicoMedico.Id, // Supondo que a entidade HistoricoMedico tenha uma propriedade Id
                    doencaId = d.DoencaId, // Supondo que Doenca tenha uma propriedade DoencaId
                    descricao = d.Descricao!,
                    medicamento = d.Medicamento!
                }).ToList();

                // Retorna as doenças encontradas com sucesso
                return CreateResponse<List<DoencaSaidaDTO>>(true, "Doenças encontradas com sucesso", doencasSaidaDTO);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao consultar doenças.");
                return CreateResponse<List<DoencaSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 10.3 - Update Doença

DoencasRepository.cs M

Repository > DoencasRepository > DoencasRepository.cs > {} TechSaude.Server.Repository.DoencasRepository > DoencasRepository > UpdateDoenca

```
namespace TechSaude.Server.Repository.DoencasRepository
    public class DoencasRepository : IDoencasRepository
        public async Task<ServerResponse<DoencaSaidaDTO>> UpdateDoenca(int id, DoencasDTO model)
        {
            try
            {
                // Verificação básica de validação do modelo
                if (string.IsNullOrEmpty(model.Descricao) || string.IsNullOrEmpty(model.Medicamento))
                {
                    return CreateResponse<DoencaSaidaDTO>(false, "Dados inválidos. Descrição e medicamento são obrigatórios.", null);
                }

                // Busca a doença pelo ID
                var doenca = await _context.Doencas.FirstOrDefaultAsync(d => d.DoencaId == id);

                if (doenca == null)
                {
                    return CreateResponse<DoencaSaidaDTO>(false, $"Nenhuma doença encontrada com o ID {id}.", null);
                }

                // Atualiza os dados da doença
                doenca.Descricao = model.Descricao;
                doenca.Medicamento = model.Medicamento;
                // Salva as mudanças no banco de dados
                await _context.SaveChangesAsync();
                // Cria o DTO para resposta
                var doencaSaidaDTO = new DoencaSaidaDTO
                {
                    historicoId = doenca.HistoricoId, // Supondo que a entidade Doenca tenha uma propriedade HistoricoId
                    doencaId = doenca.DoencaId,
                    descricao = doenca.Descricao,
                    medicamento = doenca.Medicamento
                };
                return CreateResponse<DoencaSaidaDTO>(true, "Doença atualizada com sucesso", doencaSaidaDTO);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao editar doença.");
                return CreateResponse<DoencaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

Anexo 11 - BackEnd : **Repositório de Médico**

MedicoRepository.cs M

Repository > MedicoRepository > MedicoRepository.cs > ...

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.AuthRepository;

namespace TechSaude.Server.Repository.MedicoRepository
{
    public class MedicoRepository : IMedicoRepository
    {
        private readonly AppDbContext _context;
        private readonly ILogger<MedicoRepository> _logger;
        private readonly IAuthRepository _authRepository;

        public MedicoRepository(AppDbContext context, ILogger<MedicoRepository> logger, IAuthRepository authRepository, UserManager<Usuario> userManager) {
            _context = context;
            _logger = logger;
            _authRepository = authRepository;
        }
        // RESPONSE
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = "") { return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };}
        // CRUD - CREATE, READ, UPDATE, DELETE
        public async Task<ServerResponse<List<Medico>>> GetMedicos()
        {...
        }
        public async Task<ServerResponse<Medico>> GetMedicoById(int medicoId)
        {...
        }
        public async Task<ServerResponse<Medico>> CreateMedico(RegisterMedicoDTO model)
        {...
        }
        public async Task<ServerResponse<Medico>> UpdateMedico(int medicoId, Medico model)
        {...
        }
        public async Task<ServerResponse<bool>> DeleteMedico(int medicoId)
        {...
        }
```

```
        // MÉTODOS SECUNDÁRIOS
        public async Task<ServerResponse<Consulta>> ConfirmarConsulta(int consultaId)
        {...
        }
        public async Task<ServerResponse<List<MedicoDTO>>> FiltrarMedicos(string especialidade, string localidade)
        {...
        }
        public async Task<ServerResponse<List<EspecialidadeDTO>>> GetEspecialidades()
        {...
        }
        public async Task<ServerResponse<List<LocalidadeDTO>>> GetLocalidades( string especialidade)
        {...
        }
        // FUNÇÕES
        private bool IsWeekend(DateTime date)
        {...
        }
        private bool IsWithinAllowedHours(DateTime date)
        {...
        }

    }
}
```

IMedicoRepository.cs

Repository > MedicoRepository > IMedicoRepository.cs > ...

```
using Microsoft.AspNetCore.Mvc;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.MedicoRepository
{
    public interface IMedicoRepository
    {
        // CRUD
        Task<ServerResponse<List<Medico>>> GetMedicos();
        Task<ServerResponse<Medico>> GetMedicoById(int id);
        Task<ServerResponse<Medico>> CreateMedico(RegisterMedicoDTO model);
        Task<ServerResponse<Medico>> UpdateMedico(int id, Medico model);
        Task<ServerResponse<bool>> DeleteMedico(int id);

        // Funcionalidade Específicas
        Task<ServerResponse<Consulta>> ConfirmarConsulta(int consultaId);

        Task<ServerResponse<List<EspecialidadeDTO>>> GetEspecialidades();
        Task<ServerResponse<List<LocalidadeDTO>>> GetLocalidades(string especialidade);
        Task<ServerResponse<List<MedicoDTO>>> FiltrarMedicos([FromQuery] string especialidade, [FromQuery] string localidade);
    }
}
```

Métodos Principais

GetMedicos e GetMedicoByID:

MedicoRepository.cs M

Repository > MedicoRepository > MedicoRepository.cs > TechSaude.Server.Repository.MedicoRepository > MedicoRepository > UpdateMedico

```
10  namespace TechSaude.Server.Repository.MedicoRepository
12      public class MedicoRepository : IMedicoRepository
26          public async Task<ServerResponse<List<Medico>>> GetMedicos()
27          {
28              try{
29                  var medicos = await _context.Medicos.AsNoTracking().ToListAsync();
30                  var sucesso = medicos.Any();
31                  var mensagem = sucesso ? "Solicitação realizada com sucesso." : "Nenhum médico encontrado.";
32                  return CreateResponse(sucesso, mensagem, medicos);
33              }
34              catch (Exception ex)
35              {
36                  _logger.LogError(ex, "Erro ao consultar médicos.");
37                  return CreateResponse<List<Medico>>(false, "Erro ao realizar solicitação. Tente novamente.", new List<Medico>());
38              }
39          }
40          public async Task<ServerResponse<Medico>> GetMedicoById(int medicoId)
41          {
42              try
43              {
44                  var medico = await _context.Medicos.FirstOrDefaultAsync(p => p.Id == medicoId);
45                  var sucesso = medico != null;
46                  var mensagem = sucesso ? "Médico encontrado com sucesso." : "Médico não encontrado.";
47                  return CreateResponse<Medico>(sucesso, mensagem, medico!);
48              }
49              catch (Exception ex)
50              {
51                  _logger.LogError(ex, "Erro ao consultar médico pelo ID.");
52                  return CreateResponse<Medico>(false, "Erro ao realizar solicitação. Tente novamente.", null!);
53              }
54          }
```

CreateMedico:

MedicoRepository.cs M

Repository > MedicoRepository > MedicoRepository.cs > TechSaude.Server.Repository.MedicoRepository > MedicoRepository > CreateMedico

```
10  namespace TechSaude.Server.Repository.MedicoRepository
12      public class MedicoRepository : IMedicoRepository
55          public async Task<ServerResponse<Medico>> CreateMedico(RegisterMedicoDTO model)
56          {
57              try
58              {
59                  var existingUser = await _context.Medicos.FirstOrDefaultAsync(m => m.Email == model.Email);
60                  if (existingUser != null)
61                      return CreateResponse<Medico>(false, "Já existe um usuário cadastrado com este e-mail.", null!);
62
63                  if (!model.IsValidCRM())
64                      return CreateResponse<Medico>(false, "O CRM informado é inválido.", null!);
65
66
67                  var user = new Medico
68                  {
69                      Email = model.Email!,
70                      NomeCompleto = model.NomeCompleto!,
71                      CRM = model.CRM!,
72                      Especialidade = model.Especialidade!,
73                      Local = model.LocalTrabalho!,
74                      DataCadastro = DateTime.Now,
75                      StatusUser = StatusUserEnum.Ativo,
76                      Telefone = model.Telefone!
77                      //Termo = model.Termo,
78                      //Compartilhamento = model.Compartilhamento,
79                  };
80
81                  //var result = await _userManager.CreateAsync(user, model.Senha!);
82                  await _context.Medicos.AddAsync(user);
83                  await _context.SaveChangesAsync();
84                  return CreateResponse<Medico>(true, "Medico adicionado com sucesso.", user);
85              }
86              catch (Exception ex)
87              {
88                  _logger.LogError(ex, "Erro ao registrar médico.");
89                  return CreateResponse<Medico>(false, "Erro ao processar a solicitação. Tente novamente.", null!);
90              }
91          }
```

UpdateMedico:

MedicoRepository.cs M

Repository > MedicoRepository > MedicoRepository.cs > { } TechSaude.Server.Repository.MedicoRepository > MedicoRepository > UpdateMedico

```
10  namespace TechSaude.Server.Repository.MedicoRepository
12      public class MedicoRepository : IMedicoRepository
92          public async Task<ServerResponse<Medico>> UpdateMedico(int medicoId, Medico model)
94              try
95              {
96                  var medicoExistente = await _context.Medicos.FindAsync(medicoId);
97                  if (medicoExistente == null)
98                      return CreateResponse<Medico>(false, "Médico não encontrado.", null!);
99
100                 medicoExistente.NomeCompleto = model.NomeCompleto;
101                 medicoExistente.Email = model.Email;
102                 medicoExistente.Telefone = model.Telefone;
103                 medicoExistente.Especialidade = model.Especialidade;
104                 medicoExistente.Local = model.Local;
105
106                 await _context.SaveChangesAsync();
107                 return CreateResponse(true, "Médico atualizado com sucesso.", medicoExistente);
108             }
109             catch (Exception ex)
110             {
111                 _logger.LogError(ex, "Erro ao atualizar dados do médico.");
112                 return CreateResponse<Medico>(false, "Erro ao realizar solicitação. Tente novamente.", null!);
113             }
114         }
```

DeleteMedico:

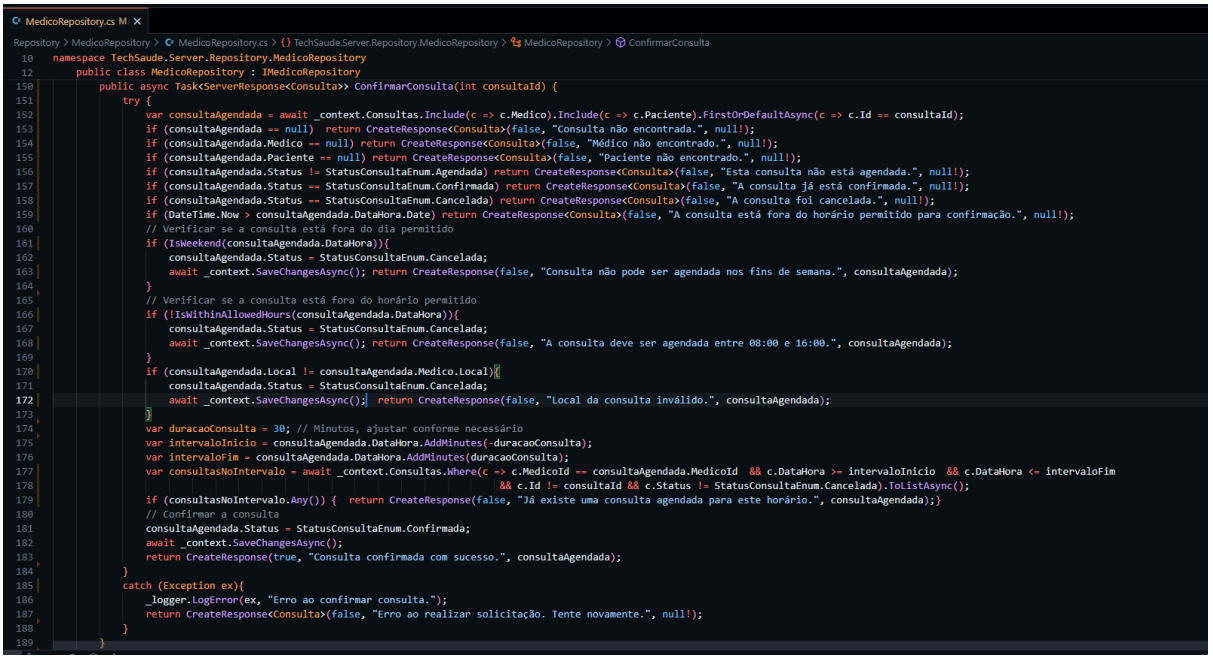
MedicoRepository.cs M

Repository > MedicoRepository > MedicoRepository.cs > { } TechSaude.Server.Repository.MedicoRepository > MedicoRepository > DeleteMedico

```
10  namespace TechSaude.Server.Repository.MedicoRepository
12      public class MedicoRepository : IMedicoRepository
114         }
115         public async Task<ServerResponse<bool>> DeleteMedico(int medicoId)
116         {
117             try
118             {
119                 var medicoResponse = await GetMedicoById(medicoId);
120                 if (!medicoResponse.Sucesso)
121                 {
122                     return CreateResponse(false, medicoResponse.Message, false);
123                 }
124
125                 //                var user = await _userManager.FindByIdAsync(medicoId.ToString());
126                 var user = await _context.Medicos.FirstOrDefaultAsync(m => m.Id == medicoId);
127                 if (user == null)
128                 {
129                     return CreateResponse(false, "Usuário associado ao médico não encontrado.", false);
130                 }
131
132                 var result = _context.Medicos.Remove(medicoResponse.Data!);
133                 await _context.SaveChangesAsync();
134
135                 //              var result = await _userManager.DeleteAsync(user);
136                 if (result == null)
137                 {
138                     return CreateResponse(false, "Erro ao deletar usuário.", false);
139                 }
140
141                 return CreateResponse(true, "Médico deletado com sucesso.", true);
142             }
143             catch (Exception ex)
144             {
145                 _logger.LogError(ex, "Erro ao deletar médico.");
146                 return CreateResponse(false, "Erro ao realizar solicitação. Tente novamente.", false);
147             }
148         }
```

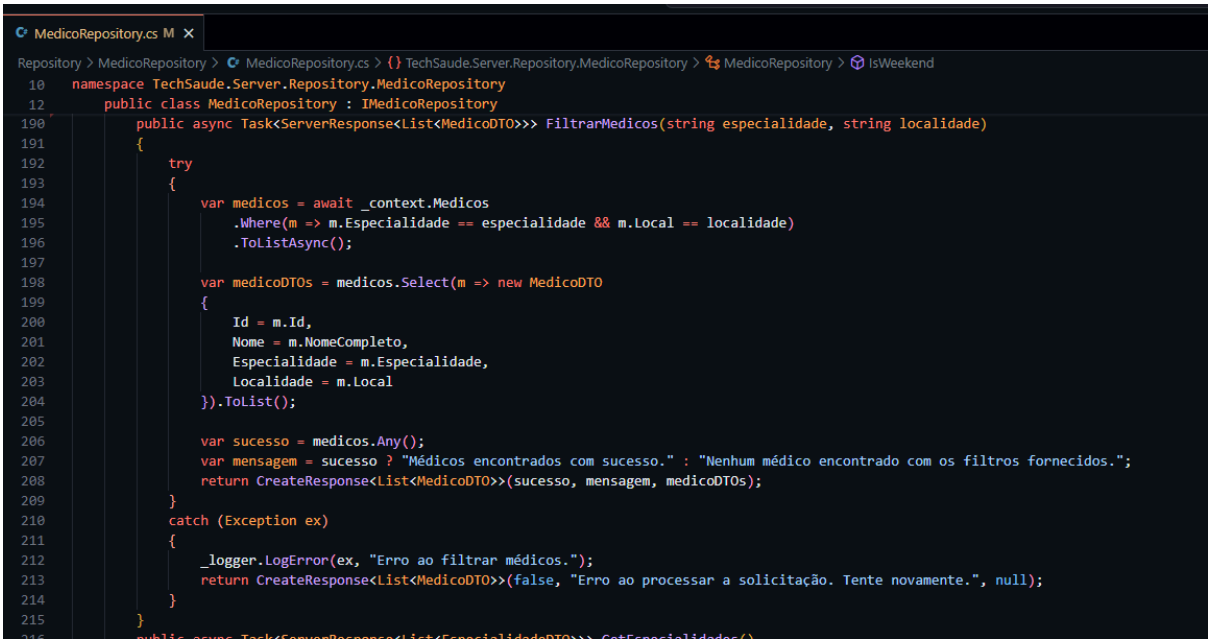
Método Secundários

ConfirmarConsulta



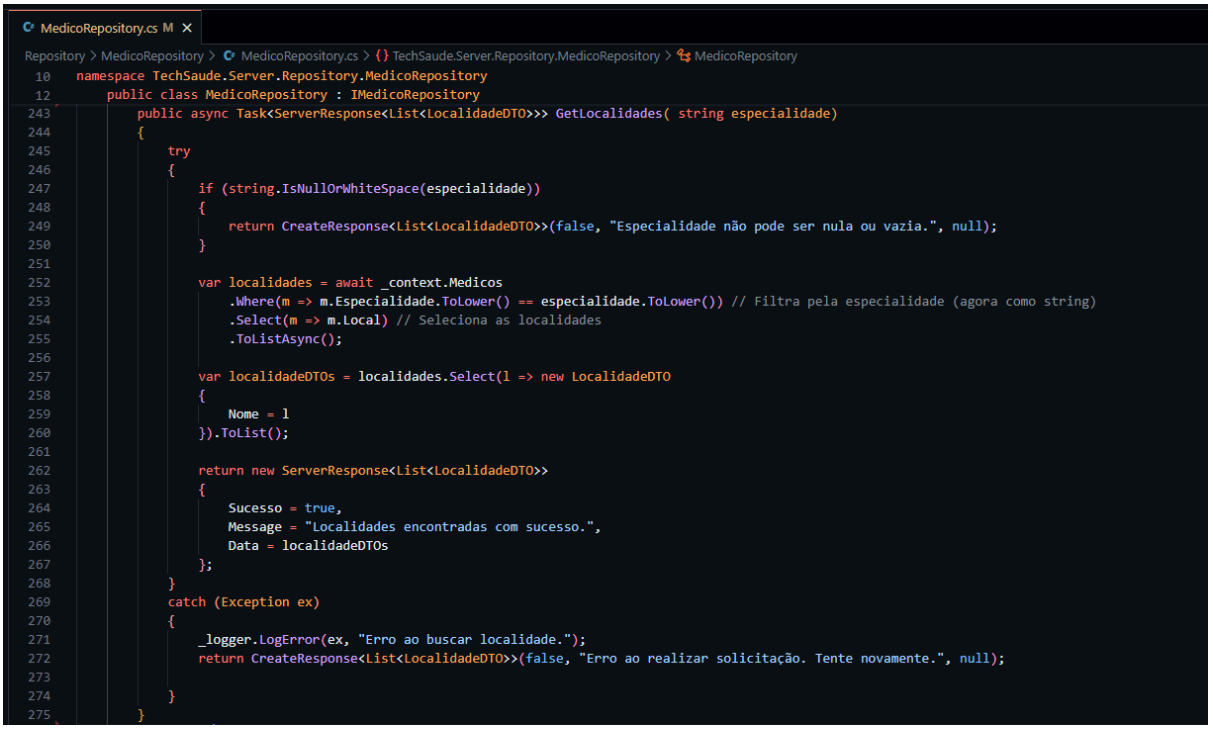
```
MedicoRepository.cs M
Repository > MedicoRepository > MedicoRepository.cs > TechSaude.Server.Repository.MedicoRepository > MedicoRepository > ConfirmarConsulta
namespace TechSaude.Server.Repository.MedicoRepository
    public class MedicoRepository : IMedicoRepository
        public async Task<ServerResponse<Consulta>> ConfirmarConsulta(int consultaId) {
            try {
                var consultaAgendada = await _context.Consultas.Include(c => c.Medico).Include(c => c.Paciente).FirstOrDefaultAsync(c => c.Id == consultaId);
                if (consultaAgendada == null)  return CreateResponse<Consulta>(false, "Consulta não encontrada.", null!);
                if (consultaAgendada.Medico == null) return CreateResponse<Consulta>(false, "Médico não encontrado.", null!);
                if (consultaAgendada.Paciente == null) return CreateResponse<Consulta>(false, "Paciente não encontrado.", null!);
                if (consultaAgendada.Status != StatusConsultaEnum.Agendada) return CreateResponse<Consulta>(false, "Esta consulta não está agendada.", null!);
                if (consultaAgendada.Status == StatusConsultaEnum.Confirmada) return CreateResponse<Consulta>(false, "A consulta já está confirmada.", null!);
                if (consultaAgendada.Status == StatusConsultaEnum.Cancelada) return CreateResponse<Consulta>(false, "A consulta foi cancelada.", null!);
                if (DateTime.Now > consultaAgendada.DataHora.Date) return CreateResponse<Consulta>(false, "A consulta está fora do horário permitido para confirmação.", null!);
                // Verificar se a consulta está fora do dia permitido
                if (IsWeekend(consultaAgendada.DataHora)){
                    consultaAgendada.Status = StatusConsultaEnum.Cancelada;
                    await _context.SaveChangesAsync(); return CreateResponse(false, "Consulta não pode ser agendada nos fins de semana.", consultaAgendada);
                }
                // Verificar se a consulta está fora do horário permitido
                if (!IsWithinAllowedHours(consultaAgendada.DataHora)){
                    consultaAgendada.Status = StatusConsultaEnum.Cancelada;
                    await _context.SaveChangesAsync(); return CreateResponse(false, "A consulta deve ser agendada entre 08:00 e 16:00.", consultaAgendada);
                }
                if (consultaAgendada.Local != consultaAgendada.Medico.Local){
                    consultaAgendada.Status = StatusConsultaEnum.Cancelada;
                    await _context.SaveChangesAsync();  return CreateResponse(false, "Local da consulta inválido.", consultaAgendada);
                }
                var duracaoConsulta = 30; // Minutos, ajustar conforme necessário
                var intervaloInicio = consultaAgendada.DataHora.AddMinutes(-duracaoConsulta);
                var intervaloFim = consultaAgendada.DataHora.AddMinutes(duracaoConsulta);
                var consultasNoIntervalo = await _context.Consultas.Where(c => c.MedicoId == consultaAgendada.MedicoId  && c.DataHora >= intervaloInicio  && c.DataHora <= intervaloFim
                                                                          && c.Id != consultaId && c.Status != StatusConsultaEnum.Cancelada).ToListAsync();
                if (consultasNoIntervalo.Any()) {  return CreateResponse(false, "Já existe uma consulta agendada para este horário.", consultaAgendada);}
                // Confirmar a consulta
                consultaAgendada.Status = StatusConsultaEnum.Confirmada;
                await _context.SaveChangesAsync();
                return CreateResponse(true, "Consulta confirmada com sucesso.", consultaAgendada);
            }
            catch (Exception ex){
                _logger.LogError(ex, "Erro ao confirmar consulta.");
                return CreateResponse<Consulta>(false, "Erro ao realizar solicitação. Tente novamente.", null!);
            }
        }
```

FiltrarMedicos



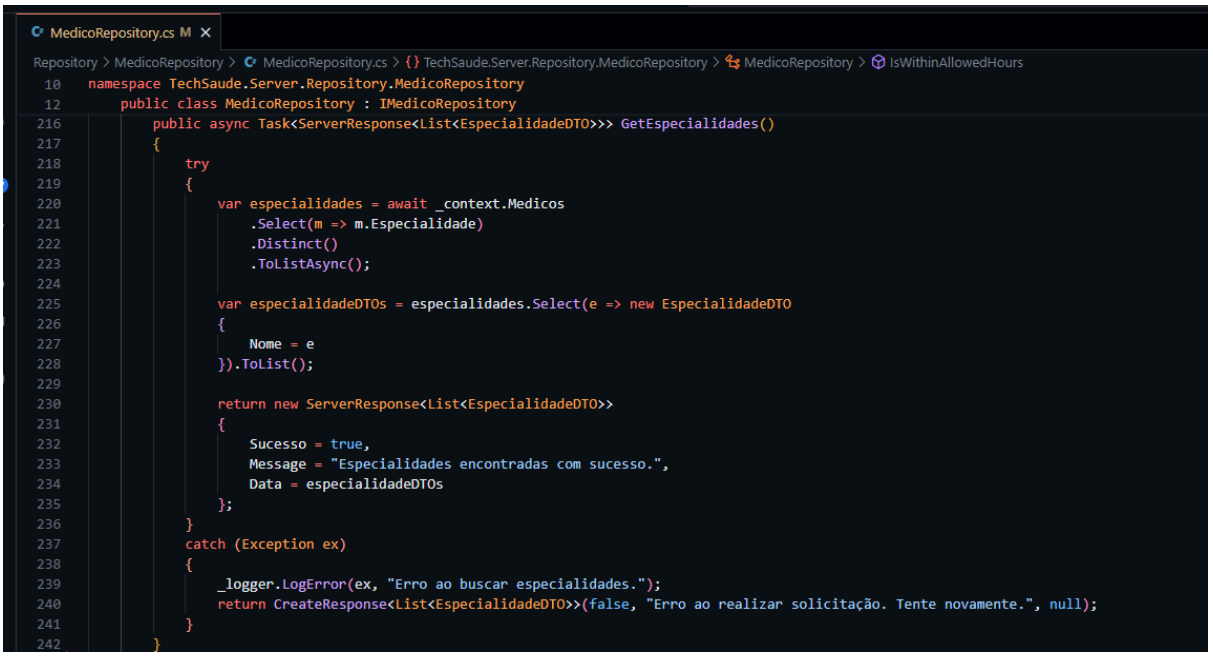
```
MedicoRepository.cs M
Repository > MedicoRepository > MedicoRepository.cs > TechSaude.Server.Repository.MedicoRepository > MedicoRepository > IsWeekend
namespace TechSaude.Server.Repository.MedicoRepository
    public class MedicoRepository : IMedicoRepository
        public async Task<ServerResponse<List<MedicoDTO>>> FiltrarMedicos(string especialidade, string localidade)
        {
            try
            {
                var medicos = await _context.Medicos
                    .Where(m => m.Especialidade == especialidade && m.Local == localidade)
                    .ToListAsync();

                var medicoDTOs = medicos.Select(m => new MedicoDTO
                {
                    Id = m.Id,
                    Nome = m.NomeCompleto,
                    Especialidade = m.Especialidade,
                    Localidade = m.Local
                }).ToList();

                var sucesso = medicos.Any();
                var mensagem = sucesso ? "Médicos encontrados com sucesso." : "Nenhum médico encontrado com os filtros fornecidos.";
                return CreateResponse<List<MedicoDTO>>(sucesso, mensagem, medicoDTOs);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao filtrar médicos.");
                return CreateResponse<List<MedicoDTO>>(false, "Erro ao processar a solicitação. Tente novamente.", null);
            }
        }
```

GetLocalidades



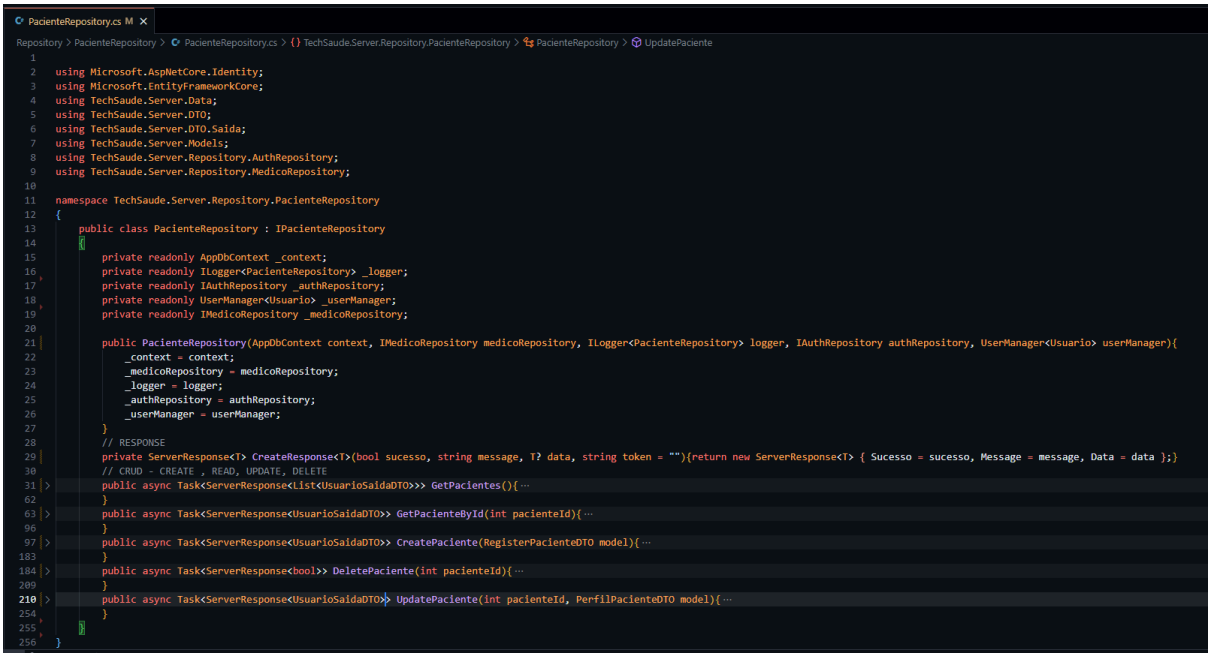
```
MedicoRepository.cs M
Repository > MedicoRepository > MedicoRepository.cs > {} TechSaude.Server.Repository.MedicoRepository > MedicoRepository
10  namespace TechSaude.Server.Repository.MedicoRepository
12      public class MedicoRepository : IMedicoRepository
243         public async Task<ServerResponse<List<LocalidadeDTO>>> GetLocalidades( string especialidade)
244         {
245             try
246             {
247                 if (string.IsNullOrWhiteSpace(especialidade))
248                 {
249                     return CreateResponse<List<LocalidadeDTO>>(false, "Especialidade não pode ser nula ou vazia.", null);
250                 }
251
252                 var localidades = await _context.Medicos
253                     .Where(m => m.Especialidade.ToLower() == especialidade.ToLower()) // Filtra pela especialidade (agora como string)
254                     .Select(m => m.Local) // Seleciona as localidades
255                     .ToListAsync();
256
257                 var localidadeDTOs = localidades.Select(l => new LocalidadeDTO
258                 {
259                     Nome = l
260                 }).ToList();
261
262                 return new ServerResponse<List<LocalidadeDTO>>
263                 {
264                     Sucesso = true,
265                     Message = "Localidades encontradas com sucesso.",
266                     Data = localidadeDTOs
267                 };
268             }
269             catch (Exception ex)
270             {
271                 _logger.LogError(ex, "Erro ao buscar localidade.");
272                 return CreateResponse<List<LocalidadeDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
273
274             }
275         }
```

GetEspecialidades

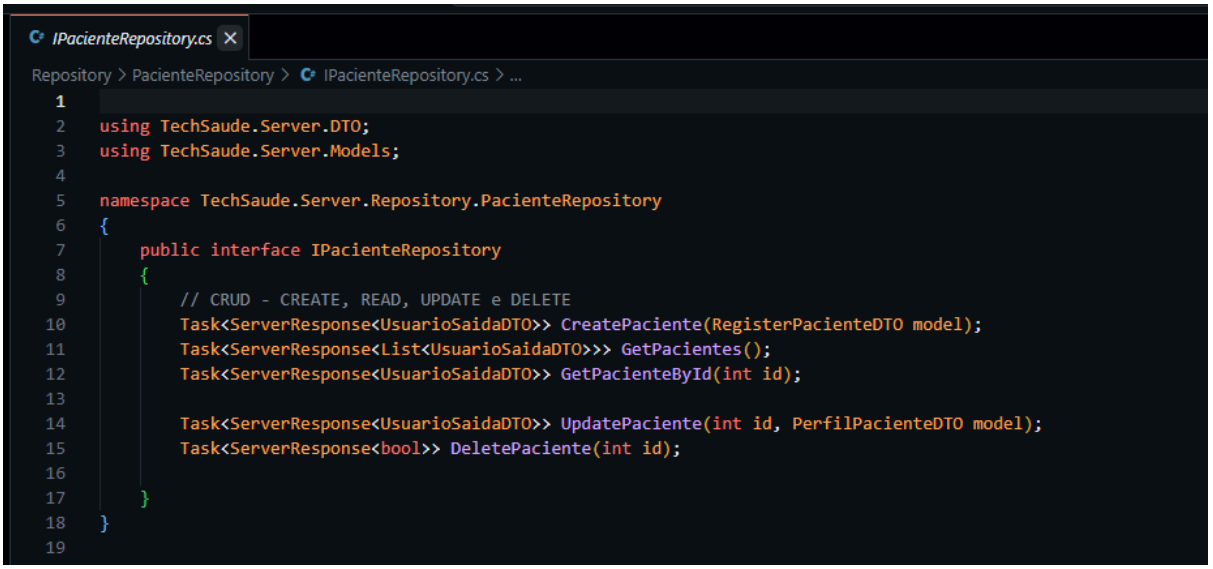


```
MedicoRepository.cs M
Repository > MedicoRepository > MedicoRepository.cs > {} TechSaude.Server.Repository.MedicoRepository > MedicoRepository > IsWithinAllowedHours
10  namespace TechSaude.Server.Repository.MedicoRepository
12      public class MedicoRepository : IMedicoRepository
216         public async Task<ServerResponse<List<EspecialidadeDTO>>> GetEspecialidades()
217         {
218             try
219             {
220                 var especialidades = await _context.Medicos
221                     .Select(m => m.Especialidade)
222                     .Distinct()
223                     .ToListAsync();
224
225                 var especialidadeDTOs = especialidades.Select(e => new EspecialidadeDTO
226                 {
227                     Nome = e
228                 }).ToList();
229
230                 return new ServerResponse<List<EspecialidadeDTO>>
231                 {
232                     Sucesso = true,
233                     Message = "Especialidades encontradas com sucesso.",
234                     Data = especialidadeDTOs
235                 };
236             }
237             catch (Exception ex)
238             {
239                 _logger.LogError(ex, "Erro ao buscar especialidades.");
240                 return CreateResponse<List<EspecialidadeDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
241             }
242         }
```

Anexo 12 - BackEnd : **Repositório de Paciente**



```
PacienteRepository.cs M
Repository > PacienteRepository > PacienteRepository.cs > {} TechSaude.Server.Repository.PacienteRepository > PacienteRepository > UpdatePaciente
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.DTO.Saida;
using TechSaude.Server.Models;
using TechSaude.Server.Repository.AuthRepository;
using TechSaude.Server.Repository.MedicoRepository;

namespace TechSaude.Server.Repository.PacienteRepository
{
    public class PacienteRepository : IPacienteRepository
    {
        private readonly AppDbContext _context;
        private readonly ILogger<PacienteRepository> _logger;
        private readonly IAuthRepository _authRepository;
        private readonly UserManager<Usuario> _userManager;
        private readonly IMedicoRepository _medicoRepository;

        public PacienteRepository(AppDbContext context, IMedicoRepository medicoRepository, ILogger<PacienteRepository> logger, IAuthRepository authRepository, UserManager<Usuario> userManager){
            _context = context;
            _medicoRepository = medicoRepository;
            _logger = logger;
            _authRepository = authRepository;
            _userManager = userManager;
        }
        // RESPONSE
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = ""){return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };}
        // CRUD - CREATE , READ, UPDATE, DELETE
        public async Task<ServerResponse<List<UsuarioSaidaDTO>>> GetPacientes(){...
        }
        public async Task<ServerResponse<UsuarioSaidaDTO>> GetPacienteById(int pacienteId){...
        }
        public async Task<ServerResponse<UsuarioSaidaDTO>> CreatePaciente(RegisterPacienteDTO model){...
        }
        public async Task<ServerResponse<bool>> DeletePaciente(int pacienteId){...
        }
        public async Task<ServerResponse<UsuarioSaidaDTO>> UpdatePaciente(int pacienteId, PerfilPacienteDTO model){...
        }
    }
}
```



```
IPacienteRepository.cs
Repository > PacienteRepository > IPacienteRepository.cs > ...
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.PacienteRepository
{
    public interface IPacienteRepository
    {
        // CRUD - CREATE, READ, UPDATE e DELETE
        Task<ServerResponse<UsuarioSaidaDTO>> CreatePaciente(RegisterPacienteDTO model);
        Task<ServerResponse<List<UsuarioSaidaDTO>>> GetPacientes();
        Task<ServerResponse<UsuarioSaidaDTO>> GetPacienteById(int id);

        Task<ServerResponse<UsuarioSaidaDTO>> UpdatePaciente(int id, PerfilPacienteDTO model);
        Task<ServerResponse<bool>> DeletePaciente(int id);

    }
}
```

GetPacientes:

PacienteRepository.cs M

Repository > PacienteRepository > PacienteRepository.cs > {} TechSaude.Server.Repository.PacienteRepository > PacienteRepository > GetPacienteById

```
11  namespace TechSaude.Server.Repository.PacienteRepository
13      public class PacienteRepository : IPacienteRepository
30          // CRUD - CREATE , READ, UPDATE, DELETE
31          public async Task<ServerResponse<List<UsuarioSaidaDTO>>> GetPacientes(){
32              try
33              {
34                  //var pacientes = await _context.Pacientes.Where(p => p.Status == StatusUsuarioEnum.Ativo).ToListAsync();
35                  var pacientes = await _context.Pacientes.Select(c => new UsuarioSaidaDTO
36                  {
37                      Id = c.Id,
38                       userName = c.UserName!,
39                       email = c.Email!,
40                       nomeCompleto = c.NomeCompleto,
41                       dataNascimento = c.DataNascimento,
42                       perfilUser = c.PerfilUser,
43                       sexo = c.Sexo,
44                       endereco = c.Endereco,
45                       telefone = c.Telefone,
46                       cns = c.CNS,
47                       convenio = c.Convenio,
48                       termo = c.Termo,
49                       compartilhamento = c.Compartilhamento
50                  }).ToListAsync();
51
52
53                  var sucesso = pacientes.Any();
54                  var mensagem = sucesso ? "Solicitação realizada com sucesso." : "Nenhum paciente encontrado.";
55                  return CreateResponse(sucesso, mensagem, pacientes);
56              }
57              catch (Exception ex)
58              {
59                  _logger.LogError(ex, "Erro ao consultar paciente.");
60                  return CreateResponse<List<UsuarioSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", new List<UsuarioSaidaDTO>());
61              }
62          }
```

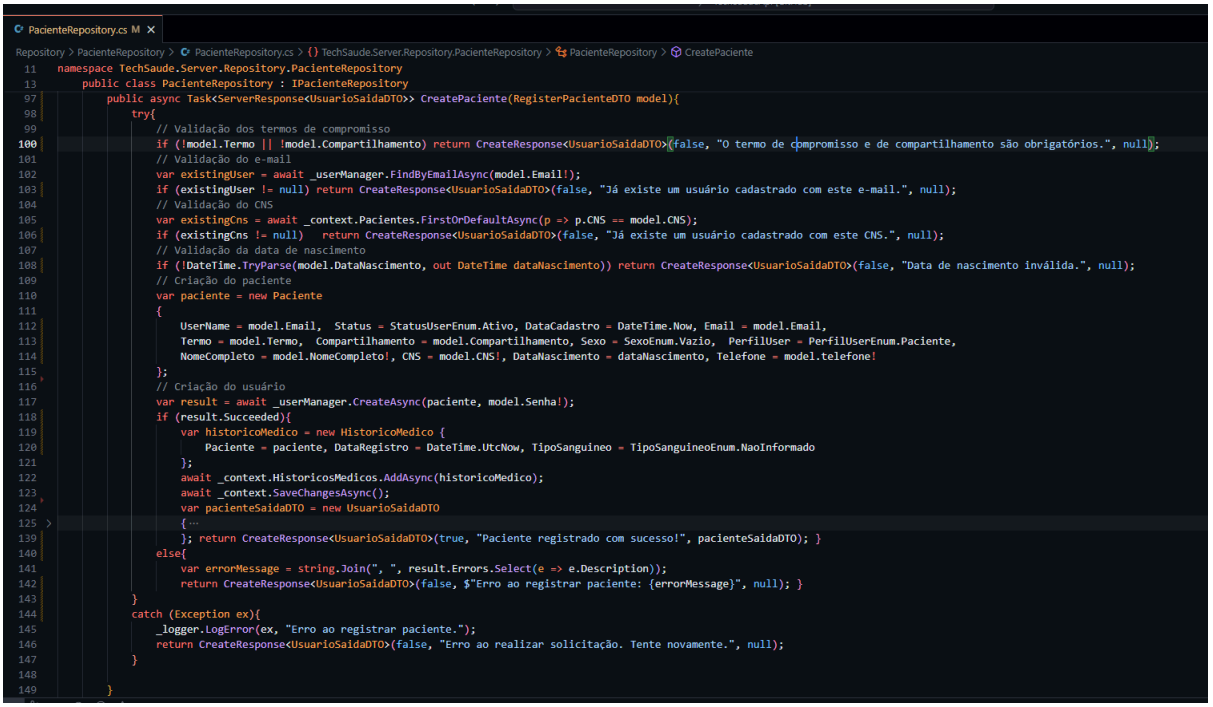
GetPacienteByID:

PacienteRepository.cs M

Repository > PacienteRepository > PacienteRepository.cs > {} TechSaude.Server.Repository.PacienteRepository > PacienteRepository > CreatePaciente

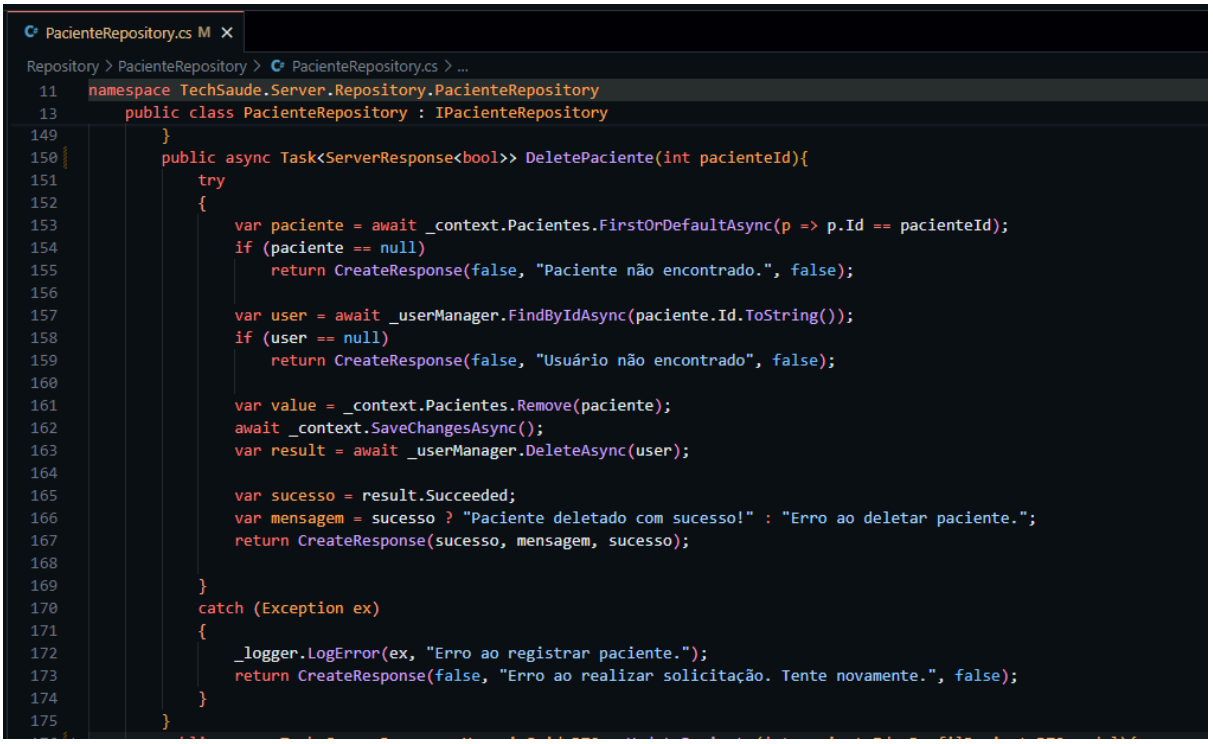
```
11  namespace TechSaude.Server.Repository.PacienteRepository
13      public class PacienteRepository : IPacienteRepository
63          public async Task<ServerResponse<UsuarioSaidaDTO>> GetPacienteById(int pacienteId){
64              try
65              {
66                  var paciente = await _context.Pacientes.FirstOrDefaultAsync(p => p.Id == pacienteId);
67
68                  if (paciente == null)
69                  {
70                      return CreateResponse<UsuarioSaidaDTO>(false, "Paciente não encontrado.", null);
71                  }
72                  var pacienteSaida = new UsuarioSaidaDTO
73                  {
74                      Id = paciente.Id,
75                      userName = paciente.UserName!,
76                      email = paciente.Email!,
77                      nomeCompleto = paciente.NomeCompleto,
78                      dataNascimento = paciente.DataNascimento,
79                      perfilUser = paciente.PerfilUser,
80                      sexo = paciente.Sexo,
81                      endereco = paciente.Endereco,
82                      telefone = paciente.Telefone,
83                      cns = paciente.CNS,
84                      convenio = paciente.Convenio,
85                      termo = paciente.Termo,
86                      compartilhamento = paciente.Compartilhamento
87                  };
88                  return CreateResponse<UsuarioSaidaDTO>(true, "Paciente  encontrado com sucesso.", pacienteSaida);
89
90              }
91              catch (Exception ex)
92              {
93                  _logger.LogError(ex, "Erro ao consultar paciente.");
94                  return CreateResponse<UsuarioSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null!);
95              }
96          }
```

CreatePaciente:



```
PacienteRepository.cs M
Repository > PacienteRepository > PacienteRepository.cs > {} TechSaude.Server.Repository.PacienteRepository > PacienteRepository > CreatePaciente
11  namespace TechSaude.Server.Repository.PacienteRepository
13      public class PacienteRepository : IPacienteRepository
97          public async Task<ServerResponse<UsuarioSaidaDTO>> CreatePaciente(RegisterPacienteDTO model){
98              try{
99                  // Validação dos termos de compromisso
100                 if (!model.Termo || !model.Compartilhamento) return CreateResponse<UsuarioSaidaDTO>(false, "O termo de compromisso e de compartilhamento são obrigatórios.", null);
101                 // Validação do e-mail
102                 var existingUser = await _userManager.FindByEmailAsync(model.Email!);
103                 if (existingUser != null) return CreateResponse<UsuarioSaidaDTO>(false, "Já existe um usuário cadastrado com este e-mail.", null);
104                 // Validação do CNS
105                 var existingCns = await _context.Pacientes.FirstOrDefaultAsync(p => p.CNS == model.CNS);
106                 if (existingCns != null)   return CreateResponse<UsuarioSaidaDTO>(false, "Já existe um usuário cadastrado com este CNS.", null);
107                 // Validação da data de nascimento
108                 if (!DateTime.TryParse(model.DataNascimento, out DateTime dataNascimento)) return CreateResponse<UsuarioSaidaDTO>(false, "Data de nascimento inválida.", null);
109                 // Criação do paciente
110                 var paciente = new Paciente
111                 {
112                     UserName = model.Email,  Status = StatusUserEnum.Ativo, DataCadastro = DateTime.Now, Email = model.Email,
113                     Termo = model.Termo,  Compartilhamento = model.Compartilhamento, Sexo = SexoEnum.Vazio,  PerfilUser = PerfilUserEnum.Paciente,
114                     NomeCompleto = model.NomeCompleto!, CNS = model.CNS!, DataNascimento = dataNascimento, Telefone = model.telefone!
115                 };
116                 // Criação do usuário
117                 var result = await _userManager.CreateAsync(paciente, model.Senha!);
118                 if (result.Succeeded){
119                     var historicoMedico = new HistoricoMedico {
120                         Paciente = paciente, DataRegistro = DateTime.UtcNow, TipoSanguineo = TipoSanguineoEnum.NaoInformado
121                     };
122                     await _context.HistoricosMedicos.AddAsync(historicoMedico);
123                     await _context.SaveChangesAsync();
124                     var pacienteSaidaDTO = new UsuarioSaidaDTO
125                     {...
139                     }; return CreateResponse<UsuarioSaidaDTO>(true, "Paciente registrado com sucesso!", pacienteSaidaDTO); }
140                 else{
141                     var errorMessage = string.Join(", ", result.Errors.Select(e => e.Description));
142                     return CreateResponse<UsuarioSaidaDTO>(false, $"Erro ao registrar paciente: {errorMessage}", null); }
143             }
144             catch (Exception ex){
145                 _logger.LogError(ex, "Erro ao registrar paciente.");
146                 return CreateResponse<UsuarioSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
147             }
148
149         }
```

DeletePaciente:



```
PacienteRepository.cs M
Repository > PacienteRepository > PacienteRepository.cs > ...
11  namespace TechSaude.Server.Repository.PacienteRepository
13      public class PacienteRepository : IPacienteRepository
149         }
150         public async Task<ServerResponse<bool>> DeletePaciente(int pacienteId){
151             try
152             {
153                 var paciente = await _context.Pacientes.FirstOrDefaultAsync(p => p.Id == pacienteId);
154                 if (paciente == null)
155                     return CreateResponse(false, "Paciente não encontrado.", false);
156
157                 var user = await _userManager.FindByIdAsync(paciente.Id.ToString());
158                 if (user == null)
159                     return CreateResponse(false, "Usuário não encontrado", false);
160
161                 var value = _context.Pacientes.Remove(paciente);
162                 await _context.SaveChangesAsync();
163                 var result = await _userManager.DeleteAsync(user);
164
165                 var sucesso = result.Succeeded;
166                 var mensagem = sucesso ? "Paciente deletado com sucesso!" : "Erro ao deletar paciente.";
167                 return CreateResponse(sucesso, mensagem, sucesso);
168
169             }
170             catch (Exception ex)
171             {
172                 _logger.LogError(ex, "Erro ao registrar paciente.");
173                 return CreateResponse(false, "Erro ao realizar solicitação. Tente novamente.", false);
174             }
175         }
```

UpdatePaciente:

PacienteRepository.cs M

Repository > PacienteRepository > PacienteRepository.cs > ...

```
11  namespace TechSaude.Server.Repository.PacienteRepository
13      public class PacienteRepository : IPacienteRepository
176         public async Task<ServerResponse<UsuarioSaidaDTO>> UpdatePaciente(int pacienteId, PerfilPacienteDTO model){
177             try{
178                 var pacienteExistente = await _context.Pacientes.FindAsync(pacienteId);
179
180                 if (pacienteExistente == null) return CreateResponse<UsuarioSaidaDTO>(false, "Paciente não encontrado.", null!);
181
182                 pacienteExistente.NomeCompleto = model.NomeCompleto!;
183                 pacienteExistente.Email = model.Email;
184                 pacienteExistente.DataNascimento = DateTime.Parse(model.DataNascimento!.ToString());
185                 pacienteExistente.Sexo = model.Sexo;
186                 pacienteExistente.Endereco = model.Endereco!;
187                 pacienteExistente.Telefone = model.Telefone!;
188                 pacienteExistente.CNS = model.CNS!;
189                 pacienteExistente.Convenio = model.Convenio!;
190                 await _context.SaveChangesAsync();
191
192                 var pacienteSaidaDTO = new UsuarioSaidaDTO{ ···
206                 };
207
208                 return CreateResponse<UsuarioSaidaDTO>(true, "Paciente atualizado com sucesso.", pacienteSaidaDTO);
209             }
210             catch (Exception ex)
211             {
212                 _logger.LogError(ex, "Erro ao atualizar dados do paciente.");
213                 return CreateResponse<UsuarioSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
214             }
215         }
216     }
217 }
218
```

VacinaRepository.cs M

Repository > VacinaRepository > VacinaRepository.cs > ...

```
using Microsoft.EntityFrameworkCore;
using TechSaude.Server.Data;
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.VacinaRepository
{
    public class VacinasRepository : IVacinaRepository
    {
        private readonly AppDbContext _context;
        private readonly ILogger<VacinasRepository> _logger;

        public VacinasRepository(AppDbContext context, ILogger<VacinasRepository> logger)
        {
            _context = context;
            _logger = logger;
        }

        // RESPONSE
        private ServerResponse<T> CreateResponse<T>(bool sucesso, string message, T? data, string token = "")
        {
            return new ServerResponse<T> { Sucesso = sucesso, Message = message, Data = data };
        }
        public async Task<ServerResponse<VacinaSaidaDTO>> AddVacina(int idPaciente, VacinaDTO model){ ···
        }
        public async Task<ServerResponse<List<VacinaSaidaDTO>>> GetVacinas(int idPaciente)
        { ···
        }
        public async Task<ServerResponse<VacinaSaidaDTO>> UpdateVacina(int idVacina, VacinaDTO model)
        { ···
        }

    }
}
```

IVacinaRepository.cs

Repository > VacinaRepository > IVacinaRepository.cs > ...

```
using TechSaude.Server.DTO;
using TechSaude.Server.Models;

namespace TechSaude.Server.Repository.VacinaRepository
{
    public interface IVacinaRepository
    {
        Task<ServerResponse<List<VacinaSaidaDTO>>> GetVacinas(int historicoId);
        Task<ServerResponse<VacinaSaidaDTO>> AddVacina(int idPaciente, VacinaDTO model);
        Task<ServerResponse<VacinaSaidaDTO>> UpdateVacina(int id, VacinaDTO model);
    }
}
```

AddVacina:

VacinaRepository.cs M

Repository > VacinaRepository > VacinaRepository.cs > {} TechSaude.Server.Repository.VacinaRepository > VacinasRepository > GetVacinas

```
namespace TechSaude.Server.Repository.VacinaRepository
    public class VacinasRepository : IVacinaRepository
        }
        public async Task<ServerResponse<VacinaSaidaDTO>> AddVacina(int idPaciente, VacinaDTO model){
            try{
                var historicoMedico = await _context.HistoricosMedicos.Include(h => h.Vacinas).FirstOrDefaultAsync(h => h.PacienteId == idPaciente);

                if (historicoMedico == null){ return CreateResponse<VacinaSaidaDTO>(false, $"Nenhum histórico médico encontrado para o paciente com ID {idPaciente}.", null);}
                // Validação básica dos dados da vacina
                if (string.IsNullOrEmpty(model.Nome) || model.Data == null){ return CreateResponse<VacinaSaidaDTO>(false, "Dados da vacina inválidos.", null);}
                if (!DateTime.TryParse(model.Data, out DateTime datavacina)) return CreateResponse<VacinaSaidaDTO>(false, "Data  Inválida.", null);
                // Criar a nova vacina
                var vacina = new Vacina
                {
                    Nome = model.Nome,Lote = model.Lote, UnidadeSaude = model.UnidadeSaude,Data = datavacina,
                };
                // Adicionar a vacina ao histórico médico
                historicoMedico.Vacinas!.Add(vacina);
                // Salvar as mudanças no banco de dados
                await _context.SaveChangesAsync();
                var vacinaSaida = new VacinaSaidaDTO
                {…
                };
                // Retornar a vacina adicionada com sucesso
                return CreateResponse<VacinaSaidaDTO>(true, "Vacina adicionada com sucesso", vacinaSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao adicionar vacina.");
                return CreateResponse<VacinaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

GetVacina:

VacinaRepository.cs M

Repository > VacinaRepository > VacinaRepository.cs > {} TechSaude.Server.Repository.VacinaRepository > VacinasRepository

```
namespace TechSaude.Server.Repository.VacinaRepository
    public class VacinasRepository : IVacinaRepository

        public async Task<ServerResponse<List<VacinaSaidaDTO>>> GetVacinas(int idPaciente)
        {
            try
            {
                var historicoMedico = await _context.HistoricosMedicos
                    .Include(h => h.Vacinas) // Incluí as vacinas associadas
                    .FirstOrDefaultAsync(h => h.PacienteId == idPaciente);

                if (historicoMedico == null)
                {
                    return CreateResponse<List<VacinaSaidaDTO>>(false, $"Nenhum histórico médico encontrado para o paciente com ID {idPaciente}.", null);
                }

                // Mapeia as vacinas associadas ao histórico médico para a lista de DTOs
                var vacinasSaida = historicoMedico.Vacinas?.Select(v => new VacinaSaidaDTO
                {
                    historicoId = historicoMedico.Id, // Supondo que você tenha essa propriedade
                    vacinaId = v.VacinaId,
                    nome = v.Nome!,
                    unidadeSaude = v.UnidadeSaude!,
                    lote = v.Lote!,
                    data = v.Data
                }).ToList();

                return CreateResponse<List<VacinaSaidaDTO>>(true, "Vacinas encontradas com sucesso", vacinasSaida);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Erro ao consultar vacina.");
                return CreateResponse<List<VacinaSaidaDTO>>(false, "Erro ao realizar solicitação. Tente novamente.", null);
            }
        }
```

UpdateVacina:

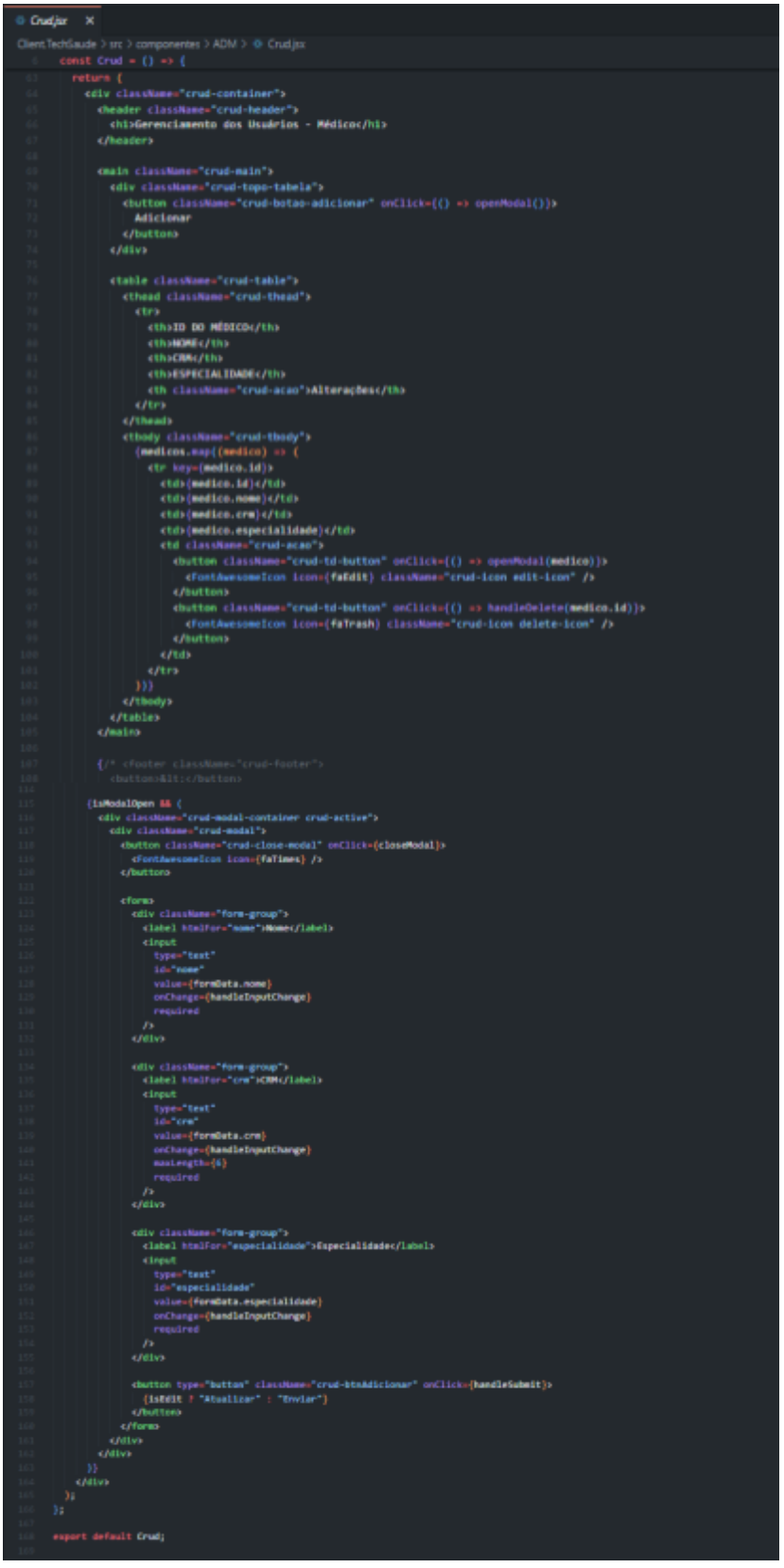
VacinaRepository.cs M

Repository > VacinaRepository > VacinaRepository.cs > TechSaude.Server.Repository.VacinaRepository > VacinasRepository > UpdateVacina

```
6   namespace TechSaude.Server.Repository.VacinaRepository
8       public class VacinasRepository : IVacinaRepository
94          public async Task<ServerResponse<VacinaSaidaDTO>> UpdateVacina(int idVacina, VacinaDTO model)
95          {
96              try
97              {
98                  var vacina = await _context.Vacinas.FirstOrDefaultAsync(v => v.VacinaId == idVacina);
99                  if (vacina == null)
100                 {
101                     return CreateResponse<VacinaSaidaDTO>(false, $"Nenhuma vacina encontrada com o ID {idVacina}.", null);
102                 }
103
104                 if (!DateTime.TryParse(model.Data, out DateTime datavacina))
105                     return CreateResponse<VacinaSaidaDTO>(false, "Data inválida.", null);
106                 // Atualiza os dados da vacina
107                 vacina.Nome = model.Nome;
108                 vacina.Lote = model.Lote;
109                 vacina.UnidadeSaude = model.UnidadeSaude;
110                 vacina.Data = datavacina;
111
112                 // Salva as mudanças no banco de dados
113                 await _context.SaveChangesAsync();
114
115                 // Mapeia a vacina atualizada para o DTO
116                 var vacinaSaida = new VacinaSaidaDTO
117                 {
118                     historicoId = vacina.HistoricoId, // Supondo que você tenha essa propriedade
119                     vacinaId = vacina.VacinaId,
120                     nome = vacina.Nome!,
121                     unidadeSaude = vacina.UnidadeSaude!,
122                     lote = vacina.Lote!,
123                     data = vacina.Data
124                 };
125
126                 return CreateResponse<VacinaSaidaDTO>(true, "Vacina atualizada com sucesso", vacinaSaida);
127             }
128             catch (Exception ex)
129             {
130                 _logger.LogError(ex, "Erro ao editar vacina.");
131                 return CreateResponse<VacinaSaidaDTO>(false, "Erro ao realizar solicitação. Tente novamente.", null);
132             }
133         }
```

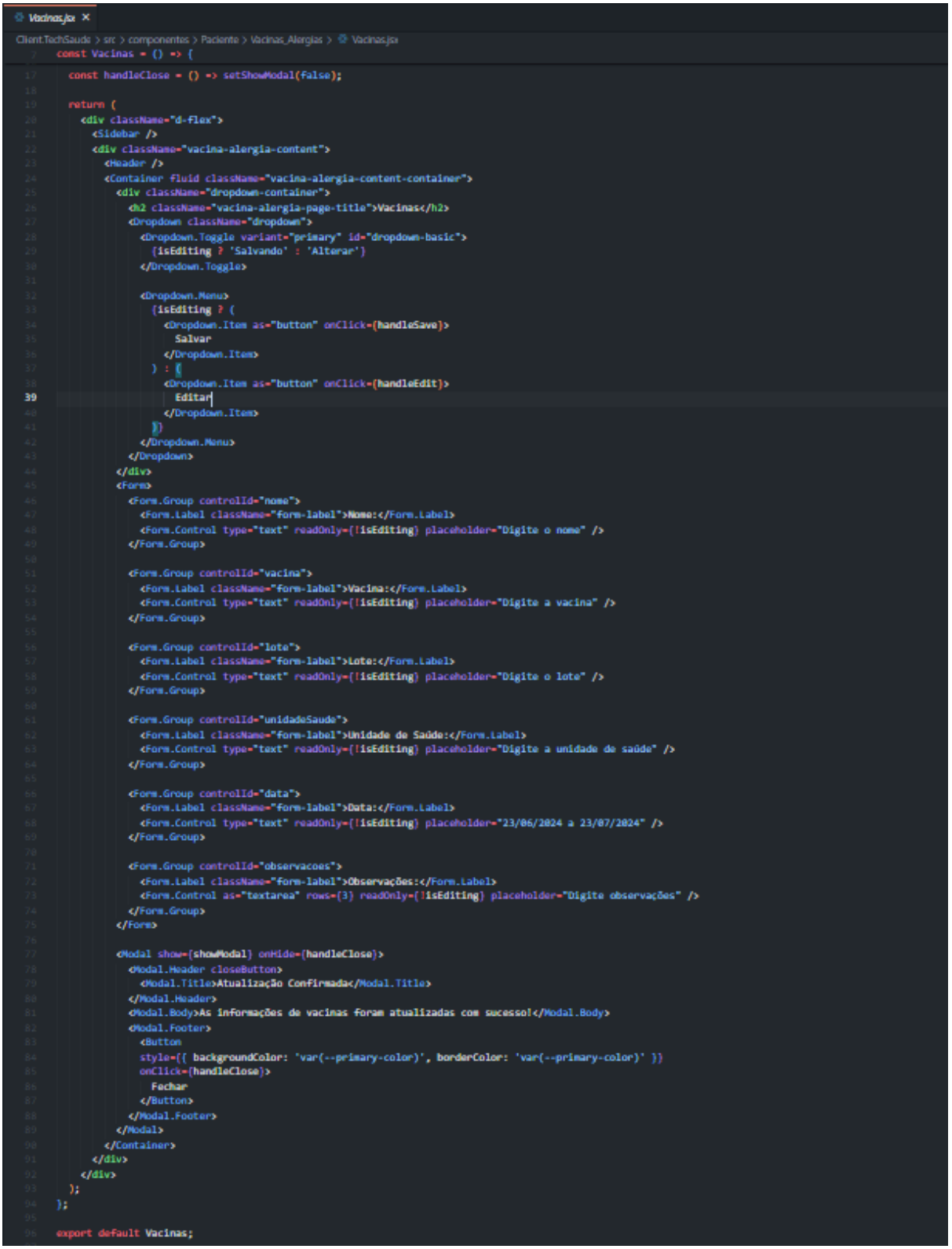
ANEXO II - Front-end:

Código da página do CRUD:



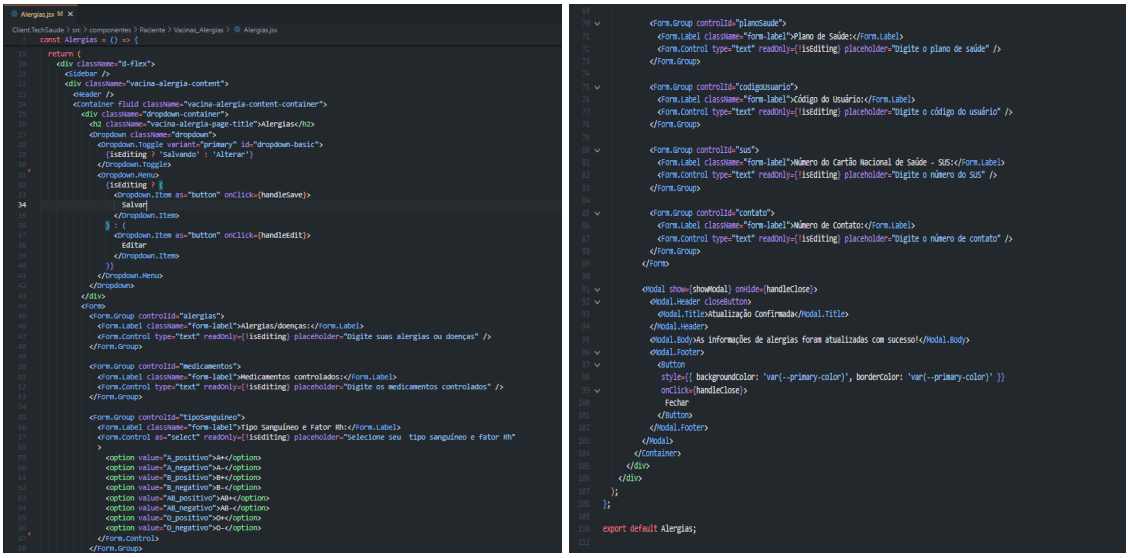
```
Crud.jsx
Client.TechSaude > src > componentes > ADM > Crud.jsx
const Crud = () => {
  return (
    <div className="crud-container">
      <header className="crud-header">
        <h1>Gerenciamento dos Usuários - Médicos</h1>
      </header>

      <main className="crud-main">
        <div className="crud-topo-tabela">
          <button className="crud-botao-adicionar" onClick={() => openModal()}>
            Adicionar
          </button>
        </div>

        <table className="crud-table">
          <thead classNome="crud-thead">
            <tr>
              <th>ID DO MÉDICO</th>
              <th>NOME</th>
              <th>CRM</th>
              <th>ESPECIALIDADE</th>
              <th classNome="crud-acao">Alterações</th>
            </tr>
          </thead>
          <tbody classNome="crud-tbody">
            {medicos.map((medico) => (
              <tr key={medico.id}>
                <td>{medico.id}</td>
                <td>{medico.nome}</td>
                <td>{medico.crm}</td>
                <td>{medico.especialidade}</td>
                <td classNome="crud-acao">
                  <button classNome="crud-td-button" onClick={() => openModal(medico)}>
                    <FontAwesomeIcon icon={faEdit} classNome="crud-icon edit-icon" />
                  </button>
                  <button classNome="crud-td-button" onClick={() => handleDelete(medico.id)}>
                    <FontAwesomeIcon icon={faTrash} classNome="crud-icon delete-icon" />
                  </button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </main>

      {/* <footer classNome="crud-footer">
        <button>Alt:</button>
      </footer> */}

      {isModalOpen && (
        <div classNome="crud-modal-container crud-active">
          <div classNome="crud-modal">
            <button classNome="crud-close-modal" onClick={closeModal}>
              <FontAwesomeIcon icon={faTimes} />
            </button>

            <form>
              <div classNome="form-group">
                <label htmlFor="nome">Nome:</label>
                <input
                  type="text"
                  id="nome"
                  value={formData.nome}
                  onChange={handleInputChange}
                  required
                />
              </div>

              <div classNome="form-group">
                <label htmlFor="crm">CRM:</label>
                <input
                  type="text"
                  id="crm"
                  value={formData.crm}
                  onChange={handleInputChange}
                  maxLength={6}
                  required
                />
              </div>

              <div classNome="form-group">
                <label htmlFor="especialidade">Especialidade:</label>
                <input
                  type="text"
                  id="especialidade"
                  value={formData.especialidade}
                  onChange={handleInputChange}
                  required
                />
              </div>

              <button type="button" classNome="crud-btnAdicionar" onClick={handleSubmit}>
                {isEdit ? "Atualizar" : "Enviar"}
              </button>
            </form>
          </div>
        </div>
      )}
    </div>
  );
};

export default Crud;
```

Código da página do ADM:



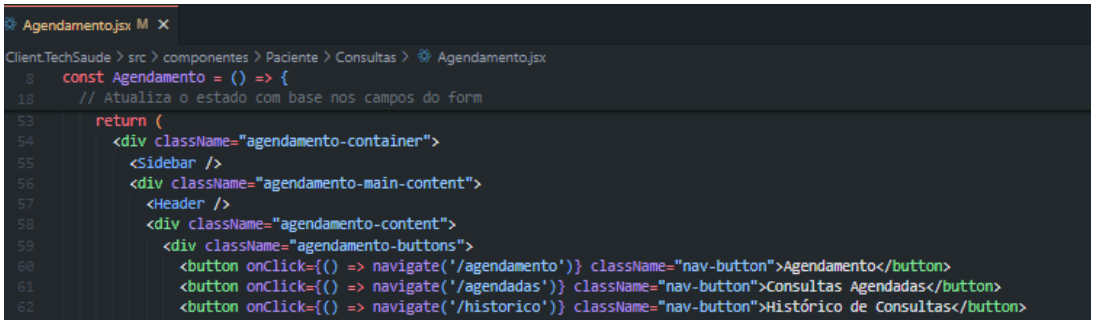
```
Vacinas.jsx
ClientTechSaude > src > componentes > Paciente > Vacinas_Alergias > Vacinas.jsx
const Vacinas = () => {
  const handleClose = () => setShowModal(false);

  return (
    <div className="d-flex">
      <Sidebar />
      <div className="vacina-alergia-content">
        <Header />
        <Container fluid className="vacina-alergia-content-container">
          <div className="dropdown-container">
            <h2 className="vacina-alergia-page-title">Vacinas</h2>
            <Dropdown className="dropdown">
              <Dropdown.Toggle variant="primary" id="dropdown-basic">
                {isEditing ? 'Salvando' : 'Alterar'}
              </Dropdown.Toggle>

              <Dropdown.Menu>
                {isEditing ? (
                  <Dropdown.Item as="button" onClick={handleSave}>
                    Salvar
                  </Dropdown.Item>
                ) : (
                  <Dropdown.Item as="button" onClick={handleEdit}>
                    Editar
                  </Dropdown.Item>
                )}
              </Dropdown.Menu>
            </Dropdown>
          </div>
          <Form>
            <Form.Group controlId="nome">
              <Form.Label className="form-label">Nome:</Form.Label>
              <Form.Control type="text" readOnly={!isEditing} placeholder="Digite o nome" />
            </Form.Group>

            <Form.Group controlId="vacina">
              <Form.Label className="form-label">Vacina:</Form.Label>
              <Form.Control type="text" readOnly={!isEditing} placeholder="Digite a vacina" />
            </Form.Group>

            <Form.Group controlId="lote">
              <Form.Label className="form-label">Lote:</Form.Label>
              <Form.Control type="text" readOnly={!isEditing} placeholder="Digite o lote" />
            </Form.Group>

            <Form.Group controlId="unidadeSaude">
              <Form.Label className="form-label">Unidade de Saúde:</Form.Label>
              <Form.Control type="text" readOnly={!isEditing} placeholder="Digite a unidade de saúde" />
            </Form.Group>

            <Form.Group controlId="data">
              <Form.Label className="form-label">Data:</Form.Label>
              <Form.Control type="text" readOnly={!isEditing} placeholder="23/06/2024 a 23/07/2024" />
            </Form.Group>

            <Form.Group controlId="observacoes">
              <Form.Label className="form-label">Observações:</Form.Label>
              <Form.Control as="textarea" rows={3} readOnly={!isEditing} placeholder="Digite observações" />
            </Form.Group>
          </Form>

          <Modal show={showModal} onHide={handleClose}>
            <Modal.Header closeButton>
              <Modal.Title>Atualização Confirmada</Modal.Title>
            </Modal.Header>
            <Modal.Body>As informações de vacinas foram atualizadas com sucesso!</Modal.Body>
            <Modal.Footer>
              <Button
              style={{ backgroundColor: 'var(--primary-color)', borderColor: 'var(--primary-color)' }}
              onClick={handleClose}>
                Fechar
              </Button>
            </Modal.Footer>
          </Modal>
        </Container>
      </div>
    </div>
  );
};

export default Vacinas;
```

Código da página de Perfil:

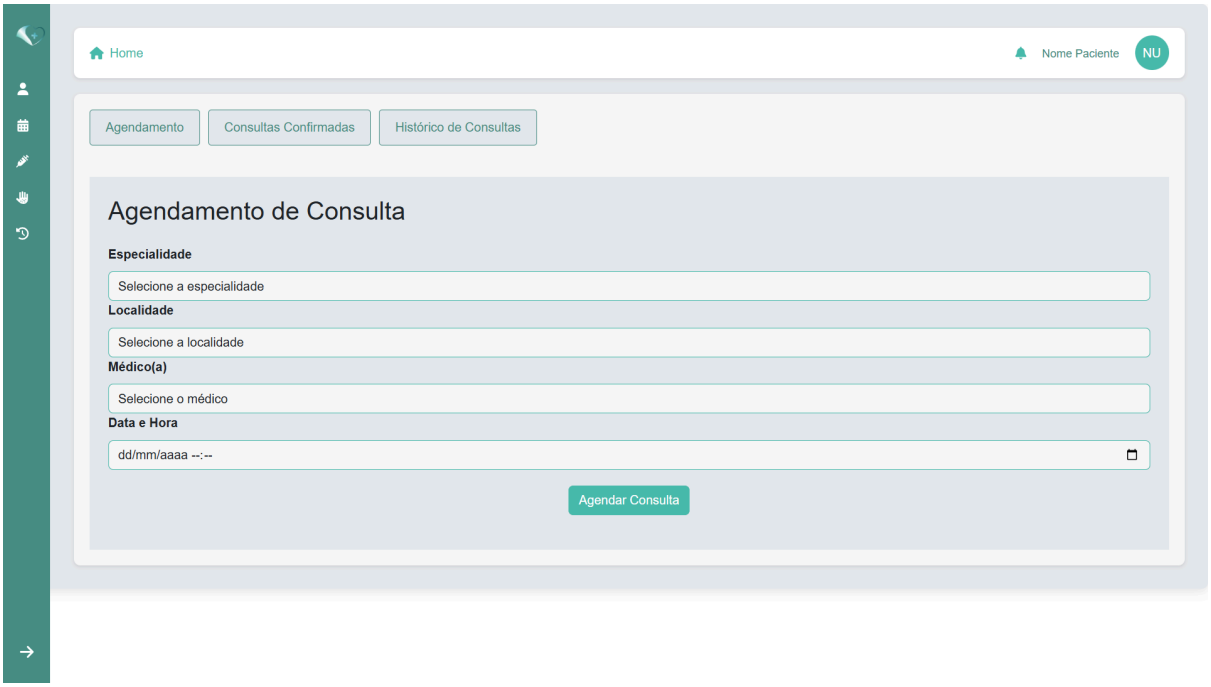


Resultado:

Código da página de agendamento:



Resultado:



Código da página de agendadas:

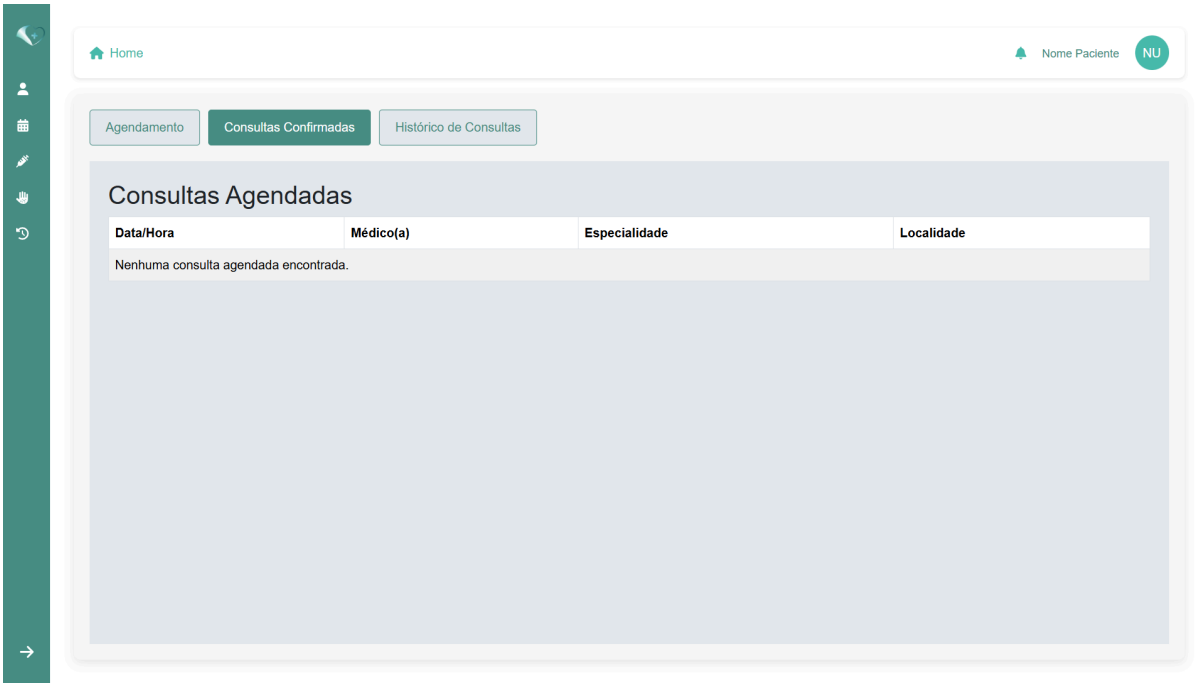
Agendadas.jsx

Client.TechSaude > src > componentes > Paciente > Consultas > Agendadas.jsx

```jsx
};

  return (
    <div className="consultas-agendadas-container">
      <Sidebar />
      <div className="consultas-agendadas-main-content">
        <Header />
        <div className="consultas-agendadas-content">
          <div className="consultas-agendadas-buttons">
            <button onClick={() => navigate('/agendamento')} className="nav-button">Agendamento</button>
            <button onClick={() => navigate('/agendadas')} className="nav-button">Consultas Agendadas</button>
            <button onClick={() => navigate('/historico')} className="nav-button">Histórico de Consultas</button>
          </div>
          <Container fluid className="consultas-agendadas-table-container p-4" style={{ backgroundColor: 'var(--background-color)' }}>
            <h2>Consultas Agendadas</h2>
            <Table striped bordered hover responsive>
              <thead>
                <tr>
                  <th>Data/Hora</th>
                  <th>Médico(a)</th>
                  <th>Especialidade</th>
                  <th>Localidade</th>
                  <th>Status</th>
                  <th>Alterações</th>
                </tr>
              </thead>
              <tbody>
                {agendadas.map((consulta) => (
                  <tr key={consulta.id}>
                    <td>{new Date(consulta.dataHora).toLocaleString()}</td>
                    <td>{consulta.medico}</td>
                    <td>{consulta.especialidade}</td>
                    <td>{consulta.localidade}</td>
                    <td>{consulta.status}</td>
                    <td>
                      <Dropdown>
                        <Dropdown.Toggle variant="success" id="dropdown-basic">
                         Alterar
                        </Dropdown.Toggle>

                        <Dropdown.Menu>
                          <Dropdown.Item onClick={() => handleAction(consulta.id, 'confirmar')}>Confirmar</Dropdown.Item>
                          <Dropdown.Item onClick={() => handleAction(consulta.id, 'cancelar')}>Cancelar</Dropdown.Item>
                        </Dropdown.Menu>
                      </Dropdown>
                    </td>
                  </tr>
                ))}
              </tbody>
            </Table>
          </Container>
        </div>
      </div>
    </div>
  );
};

export default Agendadas;
```

```
                <Form.Group controlId="medico">
                  <Form.Label>Médico(a)</Form.Label>
                  <Form.Control
                    as="select"
                    name="medico"
                    value={formData.medico}
                    onChange={handleChange}
                  >
                    <option value="">Selecione o médico</option>
                    <option value="dra-ana-silva">Dra. Ana Silva</option>
                    <option value="dra-mirella-dib">Dra. Mirella Dib Di Sessa</option>
                    <option value="dra-joao-pereira">Dra. João Pereira</option>
                  </Form.Control>
                </Form.Group>

                <Form.Group controlId="dataHora">
                  <Form.Label>Data e Hora</Form.Label>
                  <Form.Control
                    type="datetime-local"
                    name="dataHora"
                    value={formData.dataHora}
                    onChange={handleChange}
                  />
                </Form.Group>
              </Form>
            </Container>
          </div>
        </div>
      </div>
    );
};

export default Agendamento;
```

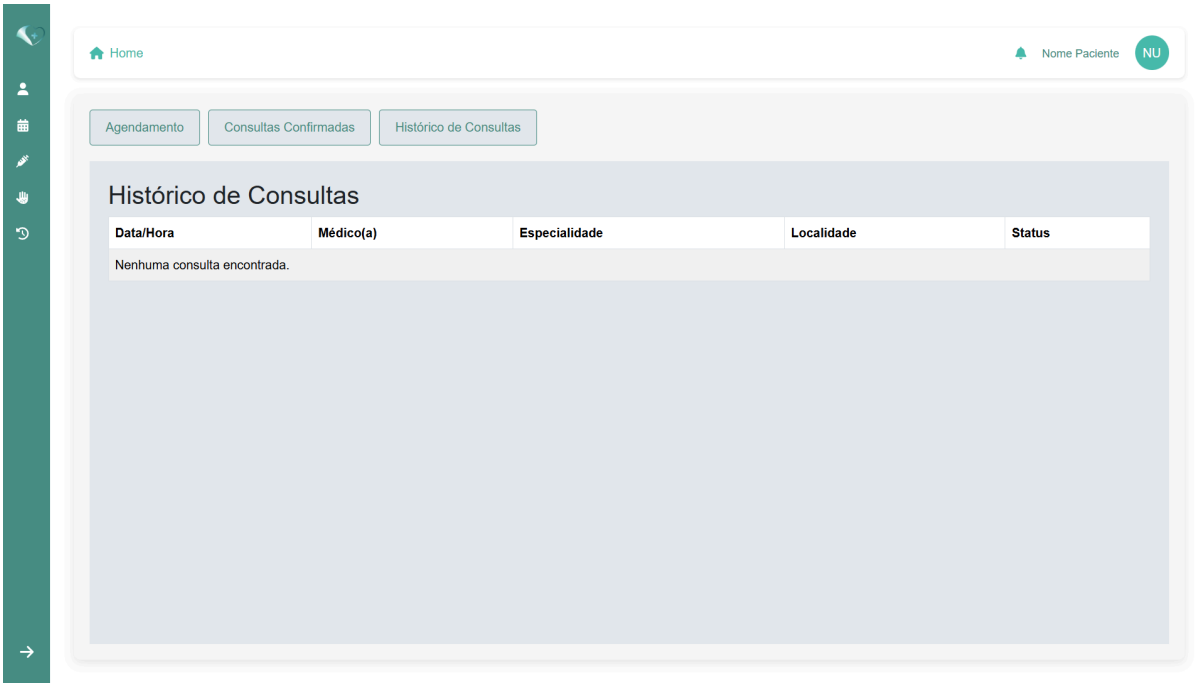
Resultado:



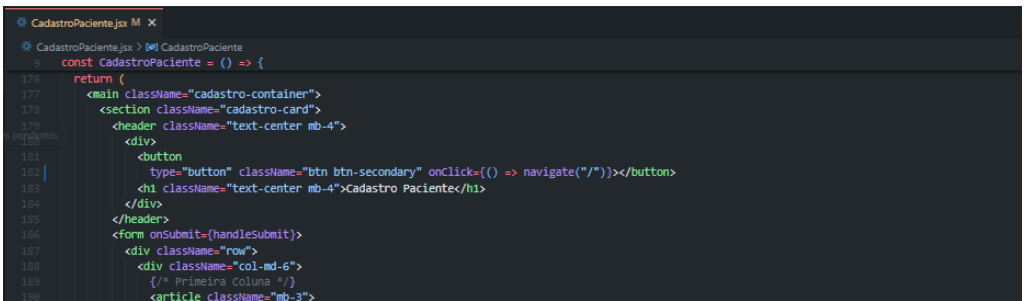
Código da página de histórico:

```jsx
    return (
      <div className="historico-consultas-container">
        <Sidebar />
        <div className="historico-consultas-main-content">
          <Header />
          <div className="historico-consultas-content">
            <div className="historico-consultas-buttons">
              <button onClick={() => navigate('/agendamento')} className="nav-button">Agendamento</button>
              <button onClick={() => navigate('/agendadas')} className="nav-button">Consultas Agendadas</button>
              <button onClick={() => navigate('/historico')} className="nav-button">Histórico de Consultas</button>
            </div>
            <Container fluid className="historico-consultas-table-container p-4" style={{ backgroundColor: 'var(--background-color)' }}>
              <h2>Histórico de Consultas</h2>
              <Table striped bordered hover responsive>
                <thead>
                  <tr>
                    <th>Data/Hora</th>
                    <th>Médico(a)</th>
                    <th>Especialidade</th>
                    <th>Localidade</th>
                    <th>Status</th>
                  </tr>
                </thead>
                <tbody>
                  {historico.map((consulta) => (
                    <tr key={consulta.id}>
                      <td>{new Date(consulta.dataHora).toLocaleString()}</td>
                      <td>{consulta.medico}</td>
                      <td>{consulta.especialidade}</td>
                      <td>{consulta.localidade}</td>
                      <td>{consulta.status}</td>
                    </tr>
                  ))}
                </tbody>
              </Table>
            </Container>
          </div>
        </div>
      </div>
    );
};

export default Historico;
```

Resultado:



Código da página de cadastro:



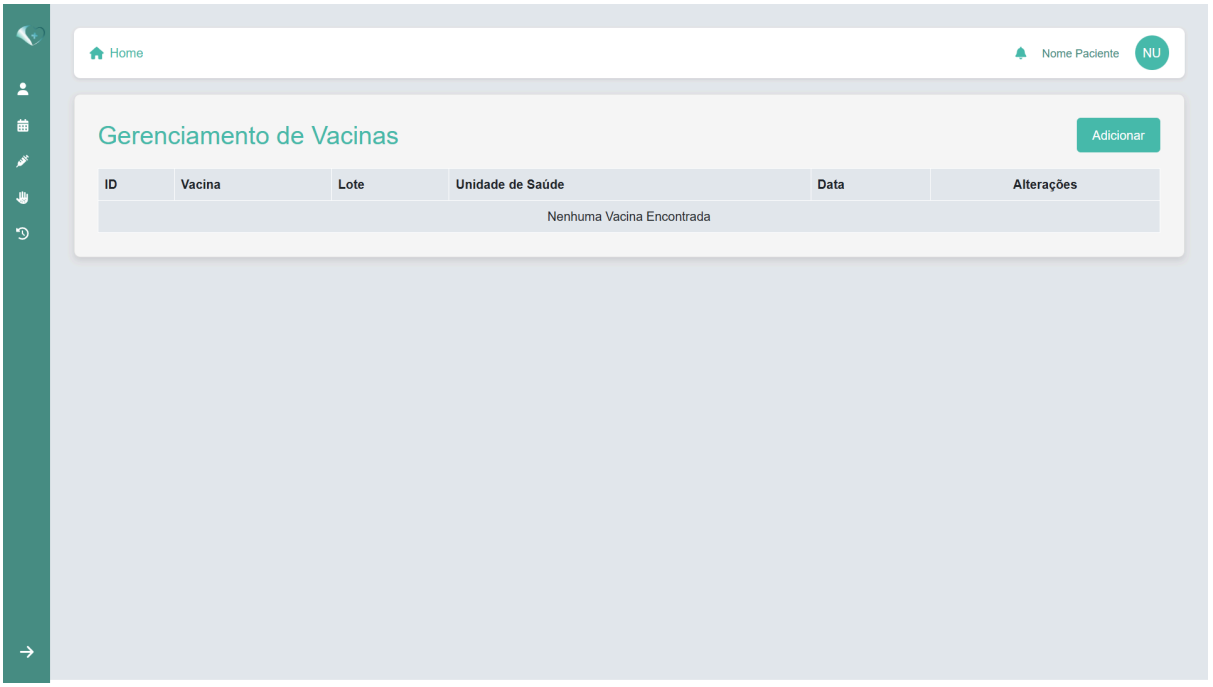
```
            </article>
            <article className="mb-3">
              <label htmlFor="senha" className="form-label" label="Senha" >
                Senha
              </label>
              <div className="input-container">
                <input type="password" id="senha" placeholder="Digite sua senha" className="form-control" value={senha}
                  onChange={(e) => setSenha(e.target.value)} aria-label="Insira a senha do paciente" aria-required="true"
                  required />
                <div className="vazio">
                  <label className="checkbox-label">
                    <input  type="checkbox" id="showSenha1" name="check" value="checkbox"
                      hidden
                      onClick={(e) => {
                        if (e.target.checked) {
                          document.getElementById("senha").type = "text";
                        } else {
                          document.getElementById("senha").type = "password";
                        }
                      }} />
                    <img src={"olhafechado.jpg"} alt="Checkbox image" />
                  </label>
                </div>
              </div>
              <ul className="senha-requisitos">
                <li>
                  {validarSenhas(senha) ? (  <i className="fas fa-check text-sucess"></i> ) : ( <i className="fas fa-times text-danger"></i> )}
                  <span>As senhas devem ser iguais.</span>
                </li>
                <li>
                  {senha.match(/[A-Z]/) ? ( <i className="fas fa-check text-success"></i> ) : ( <i className="fas fa-times text-danger"></i> )}
                  <span>Deve conter uma letra maiúscula.</span>
                </li>
                <li>
                  {senha.match(/[0-9]/) ? ( <i className="fas fa-check text-success"></i> ) : ( <i className="fas fa-times text-danger"></i> )}
                  <span>Deve conter um número.</span>
                </li>
                <li>
                  {senha.match(/[^a-zA-Z0-9]/) ? ( <i className="fas fa-check text-success"></i> ) : ( <i className="fas fa-times text-danger"></i> )}
                  <span>Deve conter um caractere especial.</span>
                </li>
              </ul>
              {erros.senha && <p className="text-danger">{erros.senha}</p>}
            </article>
            <article className="mb-3">
              <label htmlFor="confirmaSenha" className="form-label">
                Confirmar Senha
              </label>
              <div className="input-container">
                <input type="password" id="confirmaSenha" placeholder="Confirme sua senha" className="form-control" value={confirmaSenha}
                  onChange={(e) => setConfirmaSenha(e.target.value)}
                  onFocus={() => setShowSenhaMessage(true)}
                  required />
                <div className="vazio">
                  <label className="checkbox-label">
                    <input type="checkbox" id="showSenha2" name="check" value="checkbox" hidden
                      onClick={(e) => {
                        if (e.target.checked) {
```

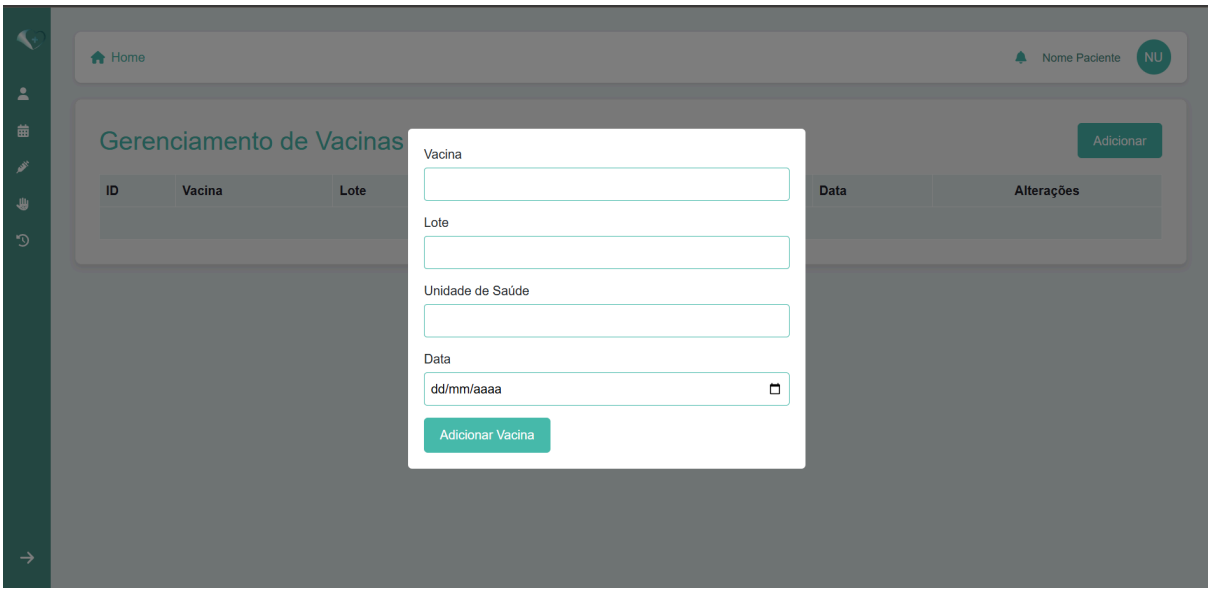
```
                          document.getElementById("confirmaSenha").type =
                            "text";
                        } else {
                          document.getElementById("confirmaSenha").type =
                            "password"; }
                      }} />
                    <img src={"olhafechado.jpg"} alt="Checkbox image" />
                  </label>
                </div>
              </div>
              {showSenhaMessage && // Adicionado condição para mostrar a mensagem
                (confirmaSenha === senha ? (
                  <p className="text-success">Senhas conferem!</p> ) : (  <p className="text-danger">Senhas não conferem.</p> ))}
            </article>
          </div>
          <div className="col-md-6">
            {/* Segunda Coluna */}
            <article className="mb-3">
              <label htmlFor="dataNascimento" className="form-label">
                Data de Nascimento
              </label>
              <input type="text" id="dataNascimento" placeholder="DD/MM/YYYY" className="form-control" value={dataNascimento}
                onChange={handleInputChange} aria-label="Data de nascimento"
                required />
              {erros.dataNascimento && ( <p className="text-danger">{erros.dataNascimento}</p> )}
            </article>
            <article className="mb-3">
              <label htmlFor="cns" className="form-label">
                CNS
              </label>
              <input type="text" id="cns" placeholder="Digite seu CNS" className="form-control" value={formData.cns}
                onChange={(e) =>
                  setFormData((prevFormData) => ({
                    ...prevFormData,
                    cns: e.target.value, })) }
                required />
              {erros.cns && <p className="text-danger">{erros.cns}</p>}
            </article>
            <article className="mb-3">
              <label htmlFor="telefone" className="form-label">
                Telefone
              </label>
              <input type="tel" id="telefone" placeholder="(00) 00000-0000" className="form-control" value={telefone}
                onChange={(e) => {
                  const telefoneFormatado = validarTelefone(e.target.value);
                  setTelefone(telefoneFormatado);
                }}
                required />
              {erros.telefone && (
                <p className="text-danger">{erros.telefone}</p> )}
            </article>
            <div className="row">
              <div className="col-md-12">
                <article className="mb-3">
                  <label className="form-label">
                    <div className="form-check">
                      <input type="checkbox" id="termo" className="form-check-input"
                      onChange={(e) =>
```

```
                        handleCheckboxChange("checkbox1", e.target.checked)}
                        aria-label="Concordo com o Termo de Uso"
                        required />
                      <label className="form-check-label" htmlFor="termo">
                        Eu li e concordo com os{" "}
                        <a href="/termos-e-condicoes" style={{ color: "var(--secondary-color)" }} >
                        Termos e Condições
                        </a>{" "} e a{" "}
                        <a href="./privacidade" style={{ color: "var(--secondary-color)" }} >
                          Política de Privacidade. </a>
                      </label>
                    </div>
                  </label>
                </article>
                <article className="mb-3">
                  <label className="form-label">
                    <div className="form-check">
                      <input type="checkbox" id="compartilhar" className="form-check-input"
                        onChange={(e) =>
                          handleCheckboxChange("checkbox2", e.target.checked) }
                        aria-label="Autorizo o compartilhamento de meus dados"
                        required />
                      <label className="form-check-label" htmlFor="compartilhamento" >
                        Eu permito e concordo com o {""}
                        <a href="./compartilhar" style={{ color: "var(--secondary-color)" }} >
                        Compartilhamento de Informações Médicas </a>{" "}
                        para o aplicativo e os profissionais da saúde.
                      </label>
                    </div>
                  </label>
                </article>
              </div>
            </div>
          </div>
        </div>
        <footer>
          <button type="submit" className="btn btn-primary" disabled={!buttonEnabled} aria-label="Cadastrar paciente" onClick={() => navigate("/perfil")}>Enviar Dados
          </button>
        </footer>
      </form>
    </section>
  </main>
 );
};

export default CadastroPaciente;
```

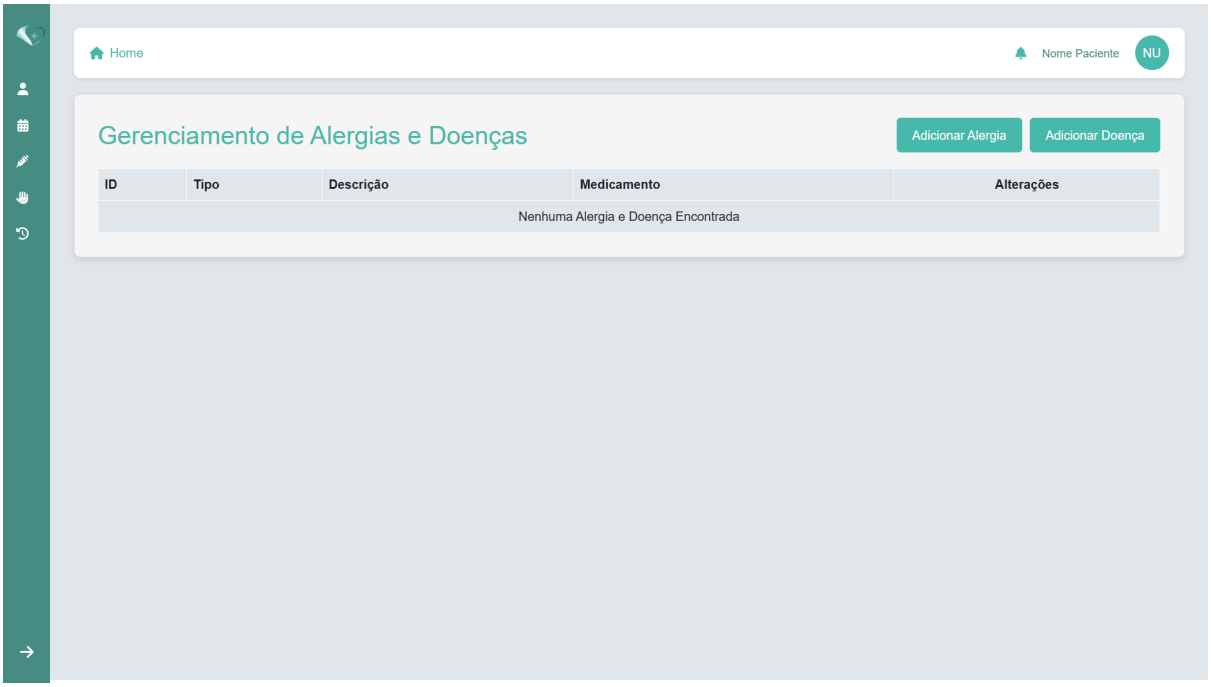
Resultado:

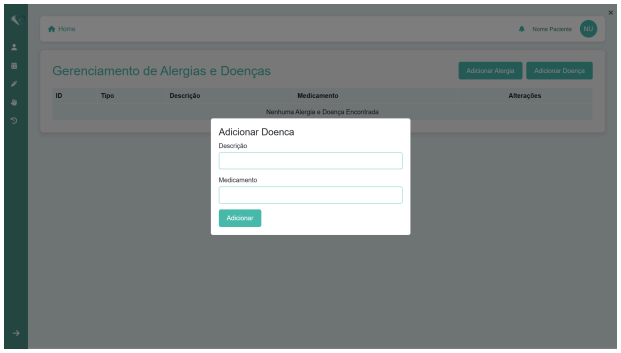
Apresentação de Tela da página de Gerenciamento de Vacinas e Pop-Up de Adição de Vacinas:

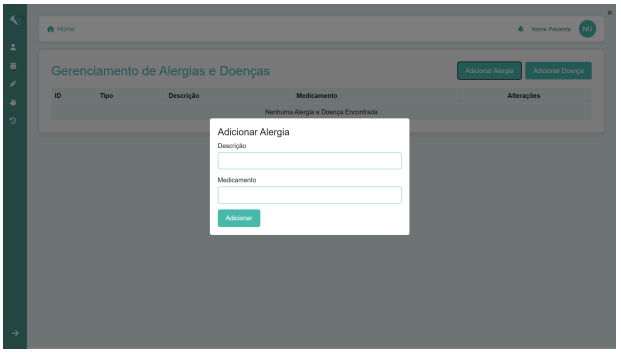




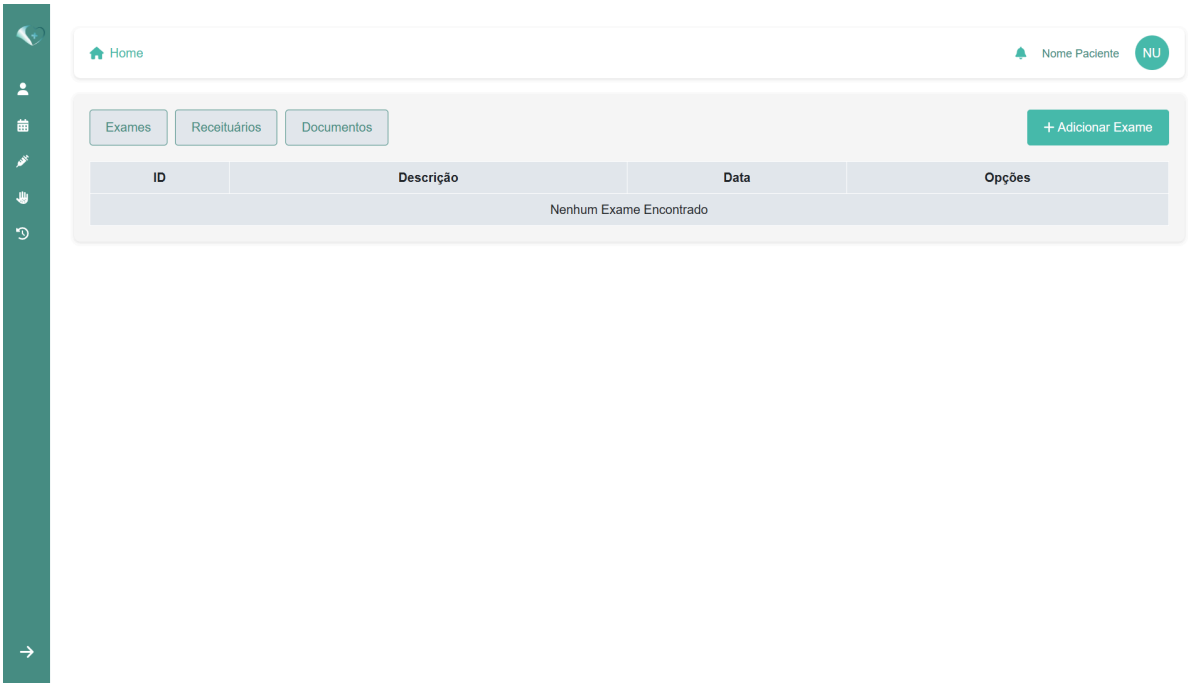
Apresentação de Tela da página de Gerenciamento de Alergias e Doenças e Pop-Up de Adição de Alergia e Adição de Doenças:

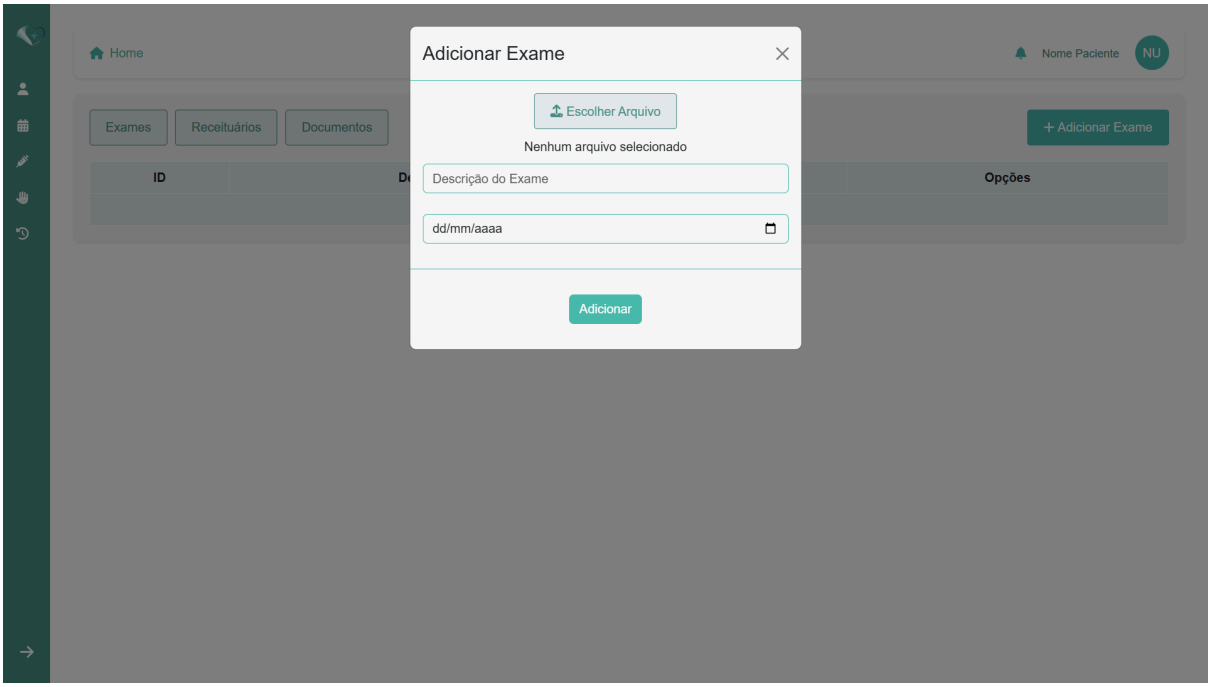




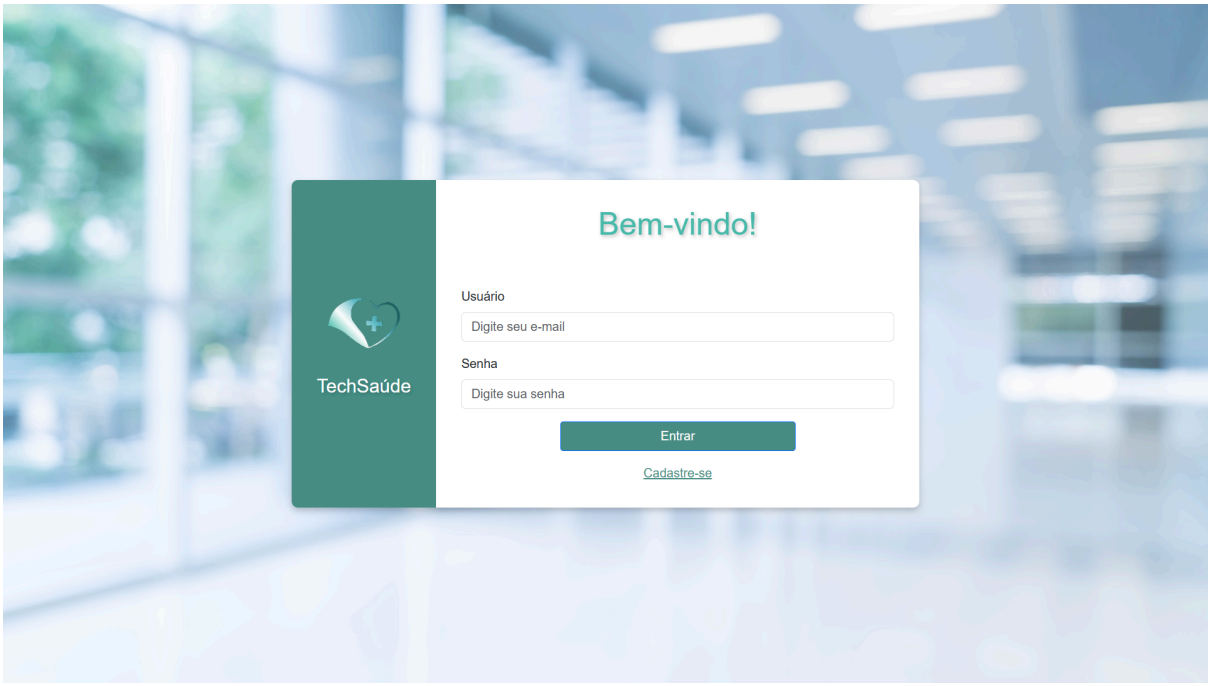


Apresentação da Tela da página de Exames e Adicionar Exames:





Resultado da página de Login:


TechSaúde
Bem-vindo!
Usuário
Digite seu e-mail
Senha
Digite sua senha
Entrar
Cadastre-se

Referências

O que é IDE (Ambiente de desenvolvimento integrado)? Disponível em: https://www.redhat.com/pt-br/topics/middleware/what-is-ide. Acesso em: 12 set. 2024.

MICROSOFT. Visual Studio Code. Disponível em: <https://code.visualstudio.com/download>. Acesso em: 13 set. 2024.

Baixar Ferramentas do Visual Studio – Instalação gratuita para Windows, Mac e Linux. Disponível em: <https://visualstudio.microsoft.com/pt-br/downloads/>. Acesso em: 16 set. 2024.

CARRER, F. O que é uma linguagem de programação e quais os tipos existem? Disponível em: <https://rockcontent.com/br/blog/linguagem-de-programacao/#:~:text=Cada%20linguagem%20de%20programa%C3%A7%C3%A3o%20%E2%80%94%20tais,entender%20quais%20comandos%20ele%20armazena.>. Acesso em: 16 set. 2024.