ANA CAROLINE FREIRE PIMENTA
JULIA OLIVEIRA TEIXEIRA
LAVÍNIA ROCHA DE LIMA
PAULO CEZAR FREIRE PIMENTA
RAYSSA FORBELONE DE FREITAS

Desenvolvimento de um Sistema Integrado para Cadastro e Anamnese no SUS: Otimizando o Atendimento e a Coleta de Informações Pessoais

Relatório do projeto

1. Descrição da proposta

O projeto teve como foco criar um sistema web intuitivo e eficiente, pensado para simplificar o cadastro e a gestão de pacientes, além de oferecer aos profissionais de saúde acesso facilitado às análises médicas. Entre as principais funcionalidades desenvolvidas estão: um sistema de login e cadastro para pacientes, a opção de escolha da unidade de saúde desejada, formulários de anamnese práticos e uma área dedicada para que os médicos consultem as informações enviadas pelos pacientes de forma organizada e segura.

1.2 Ferramentas utilizadas

Para a construção do sistema, foram empregadas tecnologias modernas e robustas. No front-end, o ReactJS foi utilizado como base para desenvolver uma interface de usuário interativa e responsiva, enquanto o NextJS proporcionou benefícios como renderização do lado do servidor (Server-Side Rendering - SSR) e maior otimização no carregamento das páginas. O back-end foi implementado com NodeJS, garantindo uma estrutura lógica sólida e escalável. O TypeScript foi integrado ao projeto, oferecendo tipagem estática ao código, o que contribuiu para a redução de erros durante o desenvolvimento e para a melhoria da segurança e da manutenção do sistema. Além disso, o Prisma foi utilizado como ferramenta de ORM, simplificando o acesso e o gerenciamento do banco de dados.

2.Gerenciamento dos dados

Uma API foi desenvolvida para viabilizar a coleta e o gerenciamento de dados. Por meio dessa solução, as informações fornecidas pelos pacientes foram armazenadas e disponibilizadas para consulta pelos médicos. A adoção dessa abordagem baseada em API garantiu uma comunicação eficiente entre o front-end e o back-end, além de possibilitar a integração de novas funcionalidades de maneira ágil e escalável.

2.2 Schema do Prisma

Com o Prisma, foi possível gerenciar os dados de login e cadastro de pacientes e médicos, as informações sobre a unidade de saúde selecionada pelos usuários e os dados fornecidos pelos pacientes para o preenchimento das anamneses.

```prisma
model Sus {
  id          Int        @id @default(autoincrement())
  nomeUnidade String
  rua         String
  numero      String
  bairro      String
  cidade      String
  estado      String
  anamneses   Anamnese[]
}

model User {
  id          Int        @id @default(autoincrement())
  nome        String
  telefone    String
  senha       String
  sexo        String
  cpf         String
  dataNasc    DateTime
  cnes        String?
  historico   Historico?
  historicoId Int
}

model Historico {
  id                Int        @id @default(autoincrement())
  pesoEmKg          Int
  alturaEmCm        String
  medicamentos      String?
  consomeAlcool     Boolean    @default(false)
  fumante           Boolean    @default(false)
  doencasCronicas   String?
  historicoFamiliar String?
  praticaExercicios Boolean
  user              User       @relation(fields: [id], references: [id])
  anamneses         Anamnese[]
}
```

```prisma
model Anamnese {
  id              Int       @id @default(autoincrement())
  sintomas        String
  rotina          String
  consultas       String
  suspeitaClinica String
  cirurgias       String
  tratamentos     String
  diagnosticos    String
  historico       Historico @relation(fields: [historicoId], references: [id])
  historicoId     Int
  sus             Sus       @relation(fields: [susId], references: [id])
  susId           Int
}
```

3. Controle dos dados

Para a inserção ou atualização dos dados armazenados em cada tabela no banco, foi desenvolvido um arquivo em TypeScript que desempenha o controle necessário.

Exemplo com os dados da tabela usuarios:

Create:

```
import { Expose, Type } from 'class-transformer';
import {
    IsString,
    IsNotEmpty,
    IsDate,
    IsOptional,
    IsNumber
} from 'class-validator';

export class CreateUsuarioDTO {
```

Get:

```

  export class GetUsuarioDTO {
```

Update:

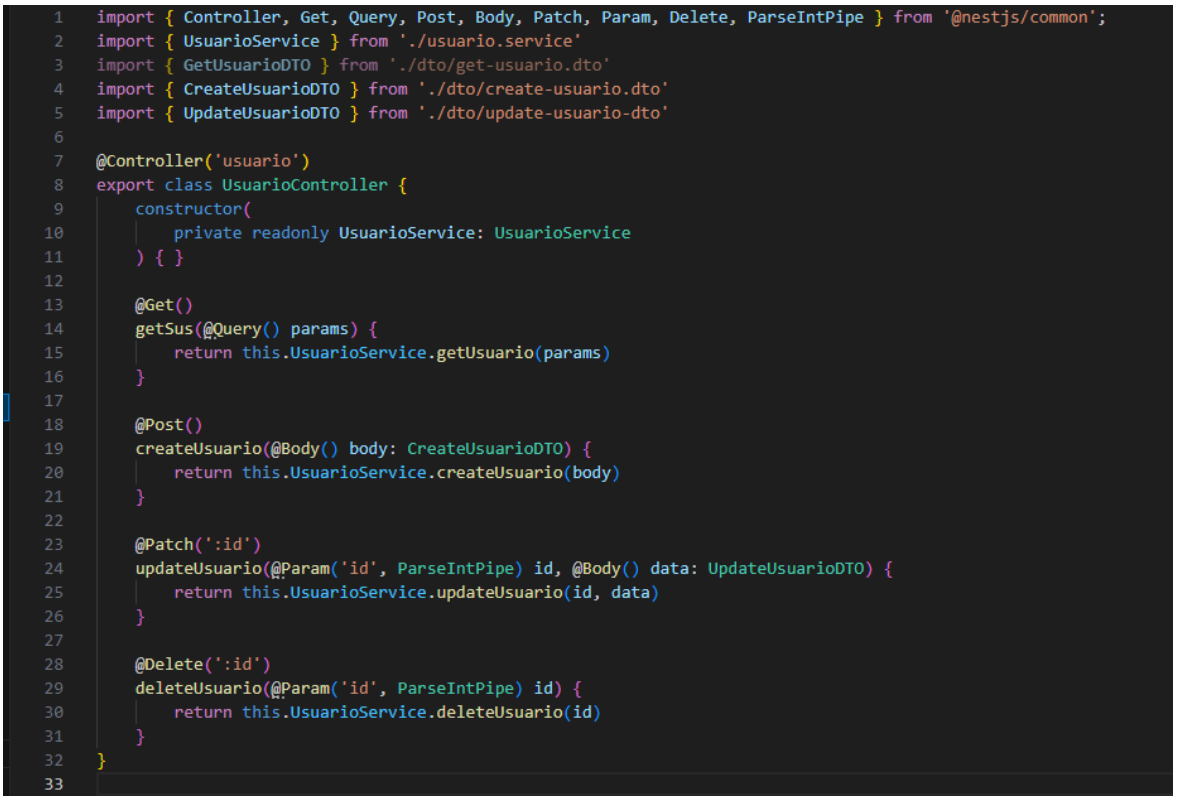
```
import { PartialType } from '@nestjs/mapped-types';
import { CreateUsuarioDTO } from './create-usuario.dto'

export class UpdateUsuarioDTO extends PartialType(CreateUsuarioDTO) {}

```

Dados da tabela:

```
export class CreateUsuarioDTO {
    @IsString()
    @IsNotEmpty()
    @Expose()
    nome: string;

    @IsString()
    @IsNotEmpty()
    @Expose()
    telefone: string;

    @IsString()
    @IsNotEmpty()
    @Expose()
    senha: string;

    @IsString()
    @IsNotEmpty()
    @Expose()
    sexo: string;

    @IsString()
    @IsNotEmpty()
    @Expose()
    cpf: string;

    @IsDate()
    @IsNotEmpty()
    @Expose()
    dataNasc: Date;

    @IsString()
    @IsOptional()
    @Expose()
    cnes?: string;

    @IsNumber()
    @IsOptional()
    @Expose()
    historicoId: number;
}
```

3.2 Construtor

Contudo, foi necessário um construtor para a atualização e a manipulação dos dados.



```
import { Controller, Get, Query, Post, Body, Patch, Param, Delete, ParseIntPipe } from '@nestjs/common';
import { UsuarioService } from './usuario.service'
import { GetUsuarioDTO } from './dto/get-usuario.dto'
import { CreateUsuarioDTO } from './dto/create-usuario.dto'
import { UpdateUsuarioDTO } from './dto/update-usuario-dto'

@Controller('usuario')
export class UsuarioController {
    constructor(
        private readonly UsuarioService: UsuarioService
    ) { }

    @Get()
    getSus(@Query() params) {
        return this.UsuarioService.getUsuario(params)
    }

    @Post()
    createUsuario(@Body() body: CreateUsuarioDTO) {
        return this.UsuarioService.createUsuario(body)
    }

    @Patch(':id')
    updateUsuario(@Param('id', ParseIntPipe) id, @Body() data: UpdateUsuarioDTO) {
        return this.UsuarioService.updateUsuario(id, data)
    }

    @Delete(':id')
    deleteUsuario(@Param('id', ParseIntPipe) id) {
        return this.UsuarioService.deleteUsuario(id)
    }
}
```

4 Ligação com o FrontEnd

Com a API em operação, os arquivos TypeScript Fetch (Que faz a busca e a comunicação dos dados com o banco) e Create (Que é responsável pela atribuição dos dados feita pelos usuários) são responsáveis por realizar a ligação com a API, facilitando a comunicação e o envio de dados entre o front-end e o back-end.

exemplo dos dados da unidade SUS:

Arquivo fecth-sus.ts:

```typescript
import { api } from '../api'

interface ISus {
  id: number;
  nomeUnidade: string
  rua: string
  numero: string
  bairro: string
  cidade: string
  estado: string
}

interface IFetchSusResponse {
  sus: ISus[];
}

export async function fetchSus(): Promise<IFetchSusResponse> {
  const response = await api.get('/sus');

  return response.data;
}
```

Arquivo create-sus.ts

```ts
import { api } from '../api'

interface ICreateSusRequest {
  nomeUnidade: string
  rua: string
  numero: string
  bairro: string
  cidade: string
  estado: string
}

interface ICreateSusResponse extends ICreateSusRequest {
  id: number
}

export async function createSus({
  nomeUnidade,
  rua,
  numero,
  estado,
  cidade,
  bairro
 }: ICreateSusRequest): Promise<ICreateSusResponse> {
  const response = await api.post('/sus', {
    nomeUnidade,
    rua,
    numero,
    estado,
    cidade,
    bairro
  });

  return response.data;
}

```

4.2 Componentes

Para o desenvolvimento do front-end em React, é necessário utilizar componentes que são integrados à página para compor a interface do usuário.

Exemplo da página da busca de unidade SUS:

Valores da tabela:

```tsx
'use client'

import { useState } from "react";
import { createSus } from "../libs/axios/modules/create-sus";

export default function CreateSus() {
  const [nomeUnidade, setNomeUnidade] = useState<string>('');
  const [cidade, setCidade] = useState<string>('');
  const [estado, setEstado] = useState<string>('');
  const [bairro, setBairro] = useState<string>('');
  const [rua, setRua] = useState<string>('');
  const [numero, setNumero] = useState<string>('');

  const handleCreateSus = async () => {
    try {
      await createSus({
        bairro,
        rua,
        numero,
        nomeUnidade,
        estado,
        cidade
      })
    } catch (error) {
      console.log(error)
    }
  }

```

Campos em que os valores serão atribuídos:

```
return(
  <div className="flex flex-col gap-4">
    <h1>Busca por unidade SUS</h1>
    <input
      placeholder="Nome da unidade"
      value={nomeUnidade}
      onChange={(text) => setNomeUnidade(text.currentTarget.value)}
      className="w-60"
    />

    <input
      placeholder="Cidade"
      value={cidade}
      onChange={(text) => setCidade(text.currentTarget.value)}
      className="w-60"
    />


    <input
      placeholder="Estado"
      value={estado}
      onChange={(text) => setEstado(text.currentTarget.value)}
      className="w-60"
    />


    <input
      placeholder="Bairro"
      value={bairro}
      onChange={(text) => setBairro(text.currentTarget.value)}
      className="w-60"
    />
```

4.3 Página com os componentes

Por fim, a página é conectada aos componentes atribuídos, permitindo a integração das funcionalidades e a estrutura desejada para a interface do usuário.

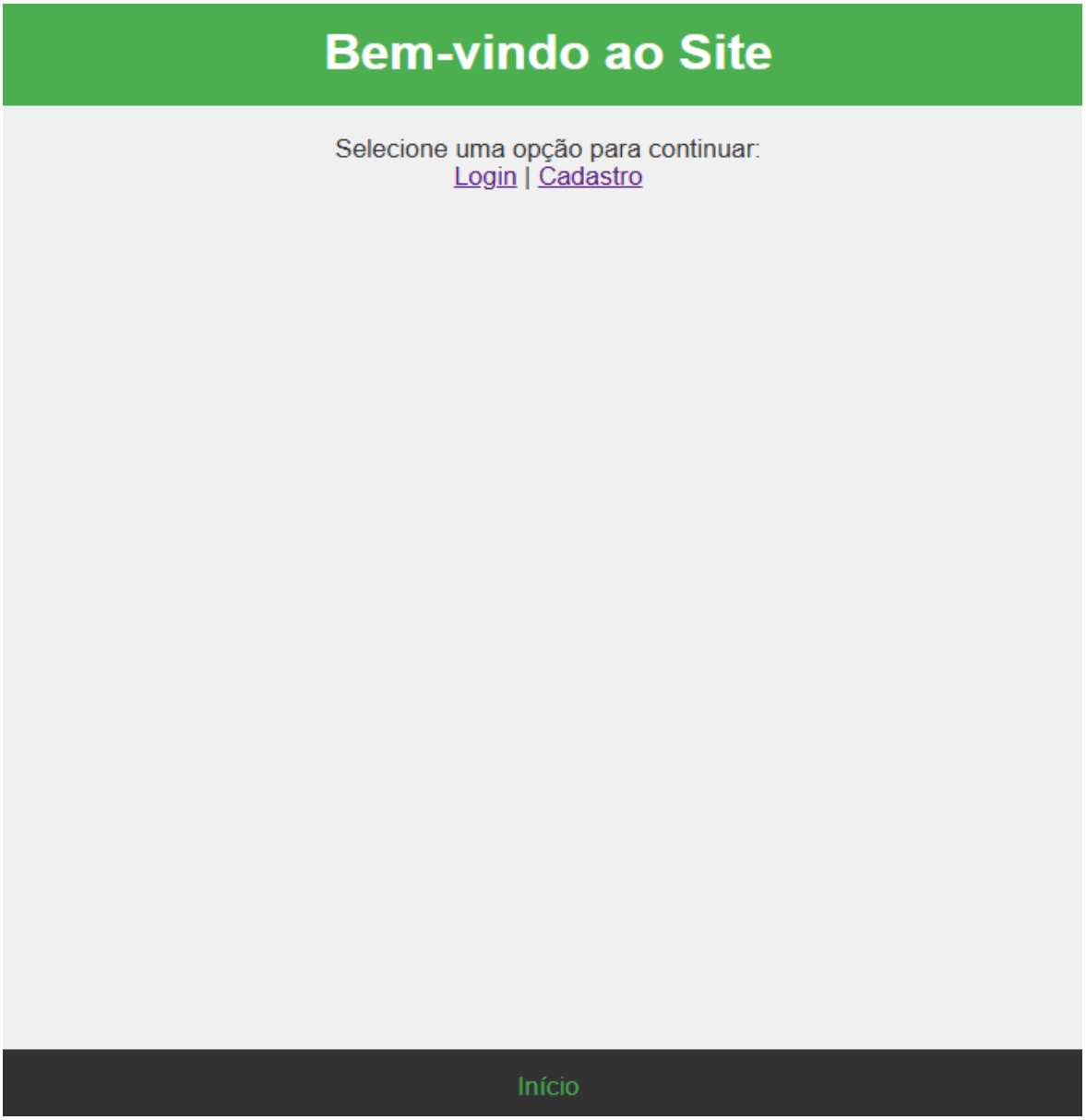
Exemplo da página da unidade SUS:

```tsx
import CreateSus from "@/src/components/CreateSus"
import { fetchSus } from "@/src/libs/axios/modules/fetch-sus"
import { use } from "react"

export default function LoginPage() {
  const susList = use(fetchSus())

  return(
    <div>
      <CreateSus />

      <h1>Lista de sus:</h1>
      {
        susList.sus.map(sus => (
          <div>
            <p>Nome da unidade: {sus.nomeUnidade}</p>
            <p>Cidade: {sus.cidade}</p>
          </div>
        ))
      }
    </div>
  )
}
```

Guia do usuário

1 Página inicial

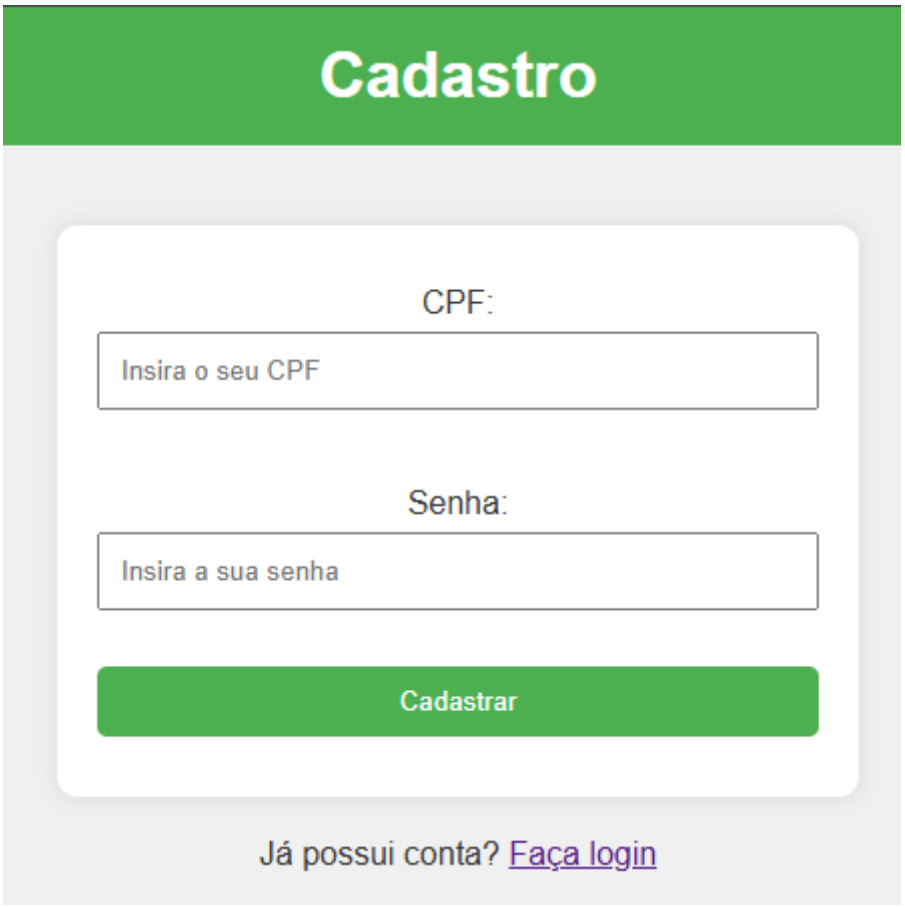
Na página inicial do site, o usuário pode optar por fazer cadastro e login caso tiver:



2 Página de Cadastro e Login

Na páginas de cadastro e login, o usuário preencherá com seu cpf e senha:

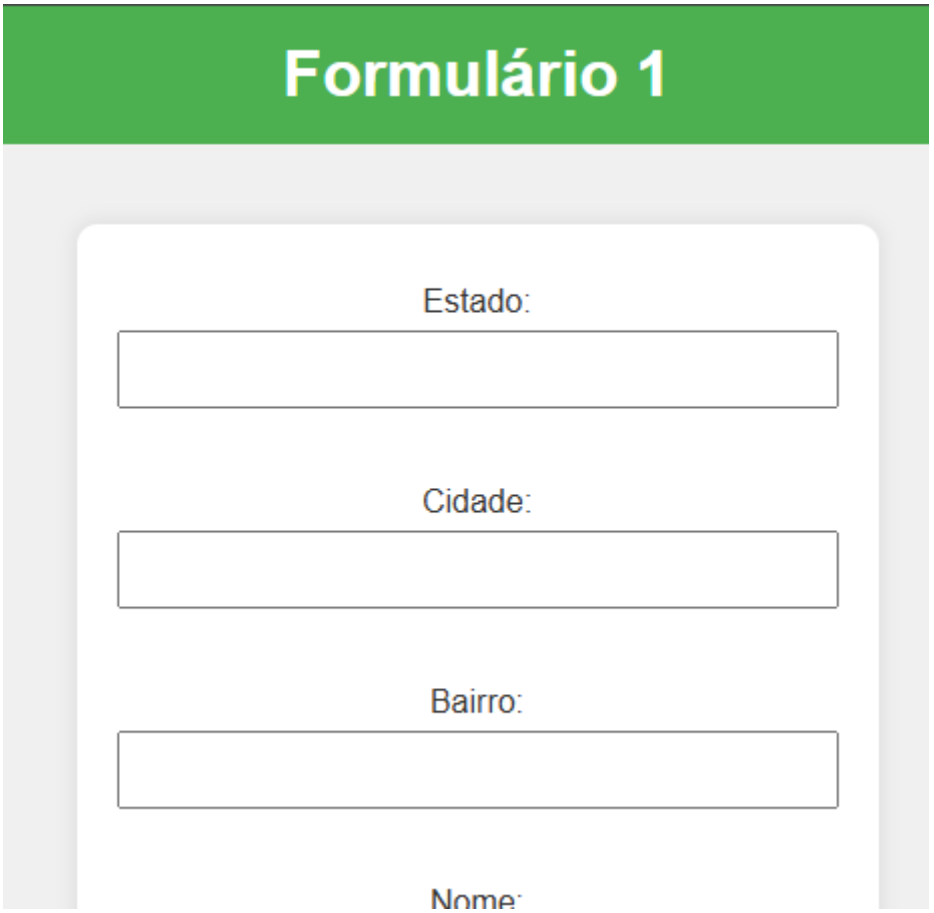
Página de cadastro:



Pagina de login:

3 Seleção de unidade:

Nesta, o usuário deverá selecionar a Unidade de Saúde desejada, procurando-a pelos seus dados de localização para prosseguir:



4 Formulário anamnese

E por fim, a página do formulário anamnese, onde o usuário deverá preencher com as informações correspondentes para o seu atendimento: