Refazendo o sistema com melhorias

Após ter configurado o ambiente e criado o projeto com base no tutorial. Abra o projeto no VSCode e siga os passos a seguir para continuar o desenvolvimento:

1. Primeiro vamos rodar a nossa aplicação, digite no terminal:
   `npm start`
   - Acesse: http://localhost:3000/ e veja como a aplicação está agora.
   - O que você está vendo é a tela padrão de um projeto React, ela está definida em **src/App.js**
2. Crie um arquivo para gerenciar as chamadas à API:
   - Crie o arquivo **src/services/api.js**
   - O arquivo ficará parecido com isso:

```
import axios from 'axios';

// Configurar a URL base da sua API
const api = axios.create({
  baseURL: 'https://angular-api.xwhost.com.br/api/', // URL real da sua API
  headers: {
    'Content-Type': 'application/json',
  },
});

export default api;
```

3. Agora vamos criar um arquivo para gerenciar as operações de clientes:
   - Crie o arquivo **src/services/ClienteService.js**
   - O arquivo ficará parecido com isso:

```
import api from './api';

// Função para buscar todos os clientes
export const getClientes = async () => {
  try {
    const response = await api.get('/clientes');
    return response.data;
  } catch (error) {
    console.error('Erro ao buscar clientes:', error);
    throw error;
  }
};

// Função para criar um cliente
export const createCliente = async (cliente) => {
  try {
    const response = await api.post('/clientes/', cliente);
    return response.data;
  } catch (error) {
```

```
    console.error('Erro ao criar cliente:', error);
    throw error;
  }
};

// Função para atualizar um cliente
export const updateCliente = async (id, clienteAtualizado) => {
  try {
    const response = await api.put(`/clientes/${id}/`, clienteAtualizado);
    return response.data;
  } catch (error) {
    console.error('Erro ao atualizar cliente:', error);
    throw error;
  }
};

// Função para deletar um cliente
export const deleteCliente = async (id) => {
  try {
    await api.delete(`/clientes/${id}`);
  } catch (error) {
    console.error('Erro ao deletar cliente:', error);
    throw error;
  }
};
```

4. Agora vamos criar um botão genérico:
   - Crie o arquivo **src/components/Button.js**
   - O arquivo ficará parecido com isso:

```
import React from 'react';
import { Button as MUIButton } from '@mui/material';

const CustomButton = ({ children, ...props }) => {
  return (
    <MUIButton {...props}>
      {children}
    </MUIButton>
  );
};

export default CustomButton;
```

Explicação:

- **Utiliza o botão do Material-UI.**
- **`children` permite que você passe o texto ou conteúdo do botão.**

5. Agora, vamos criar um componente de input que suporta máscaras.
   - Crie o arquivo **src/components/Input.js**
   - O arquivo ficará parecido com isso:

```
import React from 'react';
import InputMask from 'react-input-mask';
import { TextField } from '@mui/material';
```

```
const Input = React.forwardRef(({ label, mask, ...props }, ref) => {
  return (
    <InputMask mask={mask} {...props}>
      {(inputProps) => (
        <TextField
          {...inputProps}
          inputRef={ref}
          label={label}
          variant="outlined"
          fullWidth
          margin="normal"
        />
      )}
    </InputMask>
  );
});

export default Input;
```

Explicação:

- **InputMask**: Utilizado para aplicar a máscara (como CPF).
- **TextField**: Componente do Material-UI para campos de entrada.

6. Agora, vamos criar um componente de input que suporta máscaras.
   - Crie o arquivo **src/components/Table.js**
   - O arquivo ficará parecido com isso:

```
import React from 'react';
import { Table, TableBody, TableCell, TableContainer, TableHead, TableRow,
Paper } from '@mui/material';
import CustomButton from './Button';

const CustomTable = ({ columns, data, onEdit, onDelete }) => {
  return (
    <TableContainer component={Paper}>
      <Table>
        <TableHead>
          <TableRow>
            {columns.map((column) => (
              <TableCell key={column}>{column}</TableCell>
            ))}
            <TableCell>Ações</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {data.map((row) => (
            <TableRow key={row.id}>
              {columns.map((column) => (
                <TableCell key={column}>{row[column]}</TableCell>
              ))}
              <TableCell>
                <CustomButton onClick={() =>
onEdit(row.id)}>Editar</CustomButton>
                <CustomButton onClick={() =>
onDelete(row.id)}>Excluir</CustomButton>
              </TableCell>
```

```
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
  );
};

export default CustomTable;
```

Explicação:

- **CustomTable**: Um componente que recebe colunas, dados e funções para editar e excluir.
- Utiliza componentes de tabela do Material-UI para uma aparência moderna.

7. Agora, vamos criar a página de Listagem de Clientes
   - Crie o arquivo **src/pages/ClienteList.js**
   - O arquivo ficará parecido com isso:

```
import React, { useEffect, useState } from 'react';
import { getClientes, deleteCliente } from '../services/ClienteService';
import CustomTable from '../components/Table';
import CustomButton from '../components/Button';
import Swal from 'sweetalert2';
import { useNavigate } from 'react-router-dom';

const ClienteList = () => {
  const [clientes, setClientes] = useState([]);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchClientes = async () => {
      const response = await getClientes();
      setClientes(response);
    };
    fetchClientes();
  }, []);

  const handleDelete = async (id) => {
    const result = await Swal.fire({
      title: 'Você tem certeza que deseja deletar?',
      text: 'Este cliente será excluído definitivamente!',
      icon: 'warning',
      showCancelButton: true,
      confirmButtonText: 'Sim, excluir!',
    });

    if (result.isConfirmed) {
      await deleteCliente(id);
      setClientes(clientes.filter(cliente => cliente.id !== id));
      Swal.fire('Excluído!', 'O cliente foi excluído.', 'success');
    }
  };

  const columns = ['nome', 'cpf', 'email', ‘ativo’];
```

```
  return (
    <div>
      <h1>Lista de Clientes</h1>
      <CustomButton onClick={() => navigate('/clientes/cadastro')}>Cadastrar
Cliente</CustomButton>
      <CustomTable
        columns={columns}
        data={clientes}
        onEdit={(id) => navigate(`/clientes/editar/${id}`)}
        onDelete={handleDelete}
      />
    </div>
  );
};

export default ClienteList;
```

Explicação:

- **useEffect**: Busca os clientes quando o componente é montado.
- **handleDelete**: Função que lida com a exclusão de clientes, exibindo um alerta de confirmação.
- **CustomTable**: Renderiza a lista de clientes.

8. Agora, vamos criar a página de Cadastro e Edição de Clientes
    - Crie o arquivo **src/pages/ClienteForm.js**
    - O arquivo ficará parecido com isso:

```
import React, { useEffect } from 'react';
import { useForm } from 'react-hook-form';
import Input from '../../components/Input';
import CustomButton from '../../components/Button';
import Swal from 'sweetalert2';
import { useNavigate, useParams } from 'react-router-dom';
import { createCliente, getClientes, updateCliente } from
'../../services/ClienteService';

const ClienteForm = () => {
  const { register, handleSubmit, setValue } = useForm();
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    if (id) {
      const fetchCliente = async () => {
        const response = await getClientes();
        const cliente = response.find(c => c.id === Number(id));
        setValue('nome', cliente.nome);
        setValue('cpf', cliente.cpf);
        setValue('email', cliente.email);
        setValue('observacoes', cliente.email);

      };
```

```
      fetchCliente();
    }
  }, [id, setValue]);

  const onSubmit = async (data) => {
    try {
      if (id) {
        await updateCliente(id, data);
        Swal.fire('Sucesso!', 'Cliente atualizado com sucesso!', 'success');
      } else {
        await createCliente(data);
        Swal.fire('Sucesso!', 'Cliente cadastrado com sucesso!', 'success');
      }
      navigate('/clientes');
    } catch (error) {
      Swal.fire('Erro!', 'Ocorreu um erro ao salvar os dados.', 'error');
    }
  };

  return (
    <div className='container mt-5'>
      <h2>Cadastro clientes</h2>
      <form onSubmit={handleSubmit(onSubmit)}>
        <Input label="Nome" {...register('nome', { required: true })} />
        <Input label="CPF" mask="999.999.999-99" {...register('cpf', {
required: true })} />
        <Input label="Email" {...register('email', { required: true })} />
        <Input label="Observações" {...register('observacoes')} />
        <CustomButton variant="outlined"  type="submit">Salvar</CustomButton>
      </form>
    </div>

  );
};

export default ClienteForm;
```

Explicação:

- **useForm: Usado para gerenciar o formulário.**
- **useEffect: Carrega os dados do cliente se um ID estiver presente.**
- **onSubmit: Envia os dados do formulário para criar ou atualizar um cliente.**

9. Vamos configurar as rotas no aplicativo, vá para **src/App.js**:

```
import './App.css';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import ClienteList from './pages/cliente/ClienteList';
import ClienteForm from './pages/cliente/ClienteForm';

function App() {
  return (
    <Router>
```

```
      <Routes>
        <Route path="/" element={<ClienteList />} />
        <Route path="/clientes" element={<ClienteList />} />
        <Route path="/clientes/cadastro" element={<ClienteForm />} />
        <Route path="/clientes/editar/:id" element={<ClienteForm />} />
      </Routes>
    </Router>
  );
}

export default App;
```