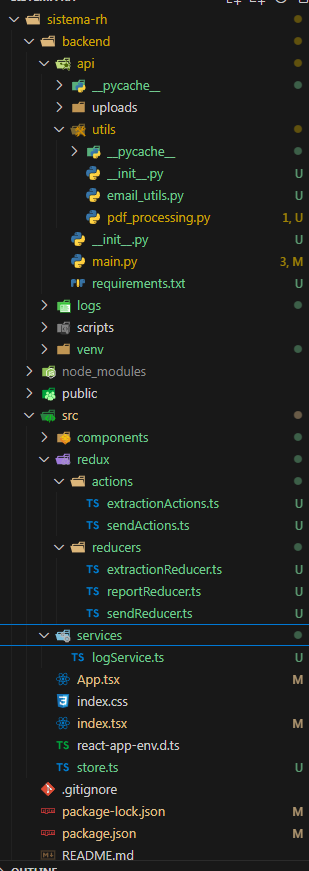
Estrutura de Arquivos



Arquivos

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\backend\api\utils\email\_utils.py"

import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

# Função para enviar notificação por e-mail em caso de erro

def enviar\_email\_notificacao(email\_destinatario, motivo\_erro):

    remetente = "seu\_email@gmail.com"

    senha = "sua\_senha"

    assunto = "Erro no Envio de Holerite"

    msg = MIMEMultipart()

    msg['From'] = remetente

    msg['To'] = email\_destinatario

    msg['Subject'] = assunto

    corpo = f"Houve um erro no envio do holerite: {motivo\_erro}"

    msg.attach(MIMEText(corpo, 'plain'))

    try:

        servidor = smtplib.SMTP('smtp.gmail.com', 587)

        servidor.starttls()

        servidor.login(remetente, senha)

        servidor.sendmail(remetente, email\_destinatario, msg.as\_string())

        servidor.quit()

        print(f"E-mail enviado para {email\_destinatario}")

    except Exception as e:

        print(f"Erro ao enviar e-mail: {str(e)}")

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\backend\api\utils\pdf\_processing.py"

import os

import io

import zipfile

import pdfplumber

from PyPDF2 import PdfReader, PdfWriter

UPLOAD\_FOLDER = 'uploads'

# Função para dividir um PDF em várias páginas

def dividir\_pdf\_em\_paginas(pdf\_file\_path):

    reader = PdfReader(pdf\_file\_path)

    num\_paginas = len(reader.pages)

    paginas\_separadas = []

    for i in range(num\_paginas):

        writer = PdfWriter()

        writer.add\_page(reader.pages[i])

        output\_filename = os.path.join(UPLOAD\_FOLDER, f"pagina\_{i+1}.pdf")

        with open(output\_filename, "wb") as output\_pdf:

            writer.write(output\_pdf)

            paginas\_separadas.append(output\_filename)

    return paginas\_separadas

# Função para renomear os arquivos com base no nome do funcionário

def extrair\_nome\_do\_pdf(caminho\_pdf):

    try:

        with pdfplumber.open(caminho\_pdf) as pdf:

            texto = ""

            for pagina in pdf.pages:

                texto += pagina.extract\_text()

            padrao\_nome = re.compile(r'Código\s+Nome do Funcionário\s+CBO\s+Departamento\s+Filial.\*\n(\d+\s+([A-Za-z\s]+))')

            resultado = padrao\_nome.search(texto)

            if resultado:

                nome = resultado.group(2).strip()

                return nome

    except Exception as e:

        print(f"Erro ao processar {caminho\_pdf}: {e}")

    return None

# Função para renomear arquivos

def renomear\_com\_funcionarios(arquivos\_paginas):

    novos\_nomes = []

    for arquivo in arquivos\_paginas:

        nome\_funcionario = extrair\_nome\_do\_pdf(arquivo)

        if nome\_funcionario:

            novo\_nome = f"{nome\_funcionario}.pdf"

        else:

            novo\_nome = os.path.basename(arquivo)  # Caso não encontre o nome, mantém o nome original

        novos\_nomes.append(novo\_nome)

    return novos\_nomes

# Função para zipar arquivos

def renomear\_e\_zipar(arquivos, novos\_nomes):

    zip\_buffer = io.BytesIO()

    with zipfile.ZipFile(zip\_buffer, 'w') as zip\_file:

        for i, arquivo in enumerate(arquivos):

            novo\_nome = novos\_nomes[i]

            zip\_file.write(arquivo, novo\_nome)

    zip\_buffer.seek(0)

    return zip\_buffer

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\backend\api\main.py"

from flask import Flask, request, jsonify, send\_file

from flask\_jwt\_extended import JWTManager, create\_access\_token, jwt\_required, get\_jwt\_identity

from flask\_cors import CORS

import os

from PyPDF2 import PdfReader, PdfWriter

import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

import datetime

import zipfile

import io

import pdfplumber

import re

app = Flask(\_\_name\_\_)

CORS(app)

# Configuração da chave secreta para o JWT

app.config['JWT\_SECRET\_KEY'] = 'sua\_chave\_secreta'

app.config['JWT\_ACCESS\_TOKEN\_EXPIRES'] = datetime.timedelta(minutes=30)  # Token expira em 30 minutos

jwt = JWTManager(app)

# Simulação de banco de dados de usuários

users\_db = {

    "admin": {"password": "1234", "role": "admin"},

    "operador": {"password": "5678", "role": "operador"},

}

# Caminho do arquivo de log para auditoria

LOG\_FILE\_PATH = '../logs/auditoria.txt'

# Caminho para armazenar arquivos temporários

UPLOAD\_FOLDER = 'uploads'

if not os.path.exists(UPLOAD\_FOLDER):

    os.makedirs(UPLOAD\_FOLDER)

# Função para gravar logs de auditoria no arquivo

def registrar\_auditoria(acao, usuario, resultado):

    if not os.path.exists('../logs'):

        os.makedirs('../logs')

    with open(LOG\_FILE\_PATH, 'a') as log\_file:

        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        log\_file.write(f"{timestamp} - {usuario}: {acao} - Resultado: {resultado}\n")

# Função para enviar notificação por e-mail em caso de erro

def enviar\_email\_notificacao(email\_destinatario, motivo\_erro):

    remetente = "seu\_email@gmail.com"

    senha = "sua\_senha"

    assunto = "Erro no Envio de Holerite"

    msg = MIMEMultipart()

    msg['From'] = remetente

    msg['To'] = email\_destinatario

    msg['Subject'] = assunto

    corpo = f"Houve um erro no envio do holerite: {motivo\_erro}"

    msg.attach(MIMEText(corpo, 'plain'))

    try:

        servidor = smtplib.SMTP('smtp.gmail.com', 587)

        servidor.starttls()

        servidor.login(remetente, senha)

        servidor.sendmail(remetente, email\_destinatario, msg.as\_string())

        servidor.quit()

        print(f"E-mail enviado para {email\_destinatario}")

    except Exception as e:

        print(f"Erro ao enviar e-mail: {str(e)}")

# Rota para login e geração de token JWT

@app.route('/api/login', methods=['POST'])

def login():

    data = request.json

    username = data.get('username')

    password = data.get('password')

    # Verifica se o usuário existe e se a senha está correta

    user = users\_db.get(username)

    if not user or user['password'] != password:

        return jsonify({"message": "Usuário ou senha incorretos"}), 401

    # Gera o token JWT

    access\_token = create\_access\_token(identity={"username": username, "role": user['role']})

    return jsonify(access\_token=access\_token), 200

# Função fictícia para simular o envio de holerite

def enviar\_holerite(holerite):

    return holerite['email'] != "erro@exemplo.com"

# Rota para envio de holerites com auditoria e notificação de erros

@app.route('/api/send-holerite', methods=['POST'])

@jwt\_required()

def send\_holerite():

    data = request.json

    usuario = get\_jwt\_identity()['username']

    success = enviar\_holerite(data)

    if success:

        registrar\_auditoria("Envio de holerite", usuario, "Sucesso")

    else:

        registrar\_auditoria("Envio de holerite", usuario, "Erro")

        enviar\_email\_notificacao(data['email'], "Erro no envio")

    return jsonify({"message": "Envio processado"}), 200

# Rota para envio de holerites em lote com auditoria e notificação de erros

@app.route('/api/send-holerites', methods=['POST'])

@jwt\_required()

def send\_holerites():

    data = request.json  # Recebe lista de holerites

    erros = []

    usuario = get\_jwt\_identity()['username']

    for holerite in data['holerites']:

        success = enviar\_holerite(holerite)

        if success:

            registrar\_auditoria("Envio de holerite em lote", usuario, "Sucesso")

        else:

            erros.append(holerite['email'])

            registrar\_auditoria("Envio de holerite em lote", usuario, "Erro")

            enviar\_email\_notificacao(holerite['email'], "Erro no envio")

    return jsonify({

        "message": "Processo de envio finalizado",

        "erros": erros

    }), 200

# Rota para acessar os logs de auditoria (somente para administradores)

@app.route('/api/logs', methods=['GET'])

@jwt\_required()

def get\_logs():

    usuario = get\_jwt\_identity()['username']

    if get\_jwt\_identity()['role'] != 'admin':

        return jsonify({"message": "Acesso negado"}), 403

    if not os.path.exists(LOG\_FILE\_PATH):

        return jsonify({"message": "Nenhum log encontrado"}), 404

    with open(LOG\_FILE\_PATH, 'r') as log\_file:

        logs = log\_file.readlines()

    registrar\_auditoria("Consulta de logs", usuario, "Sucesso")

    return jsonify(logs=logs), 200

# Função para dividir um PDF em várias páginas

def dividir\_pdf\_em\_paginas(pdf\_file\_path):

    reader = PdfReader(pdf\_file\_path)

    num\_paginas = len(reader.pages)

    paginas\_separadas = []

    for i in range(num\_paginas):

        writer = PdfWriter()

        writer.add\_page(reader.pages[i])

        output\_filename = f"{UPLOAD\_FOLDER}/pagina\_{i+1}.pdf"

        with open(output\_filename, "wb") as output\_pdf:

            writer.write(output\_pdf)

            paginas\_separadas.append(output\_filename)

    return paginas\_separadas

# Função para renomear e zipar arquivos

def renomear\_e\_zipar(arquivos, novos\_nomes):

    zip\_buffer = io.BytesIO()  # Cria um buffer para o arquivo zip

    with zipfile.ZipFile(zip\_buffer, 'w') as zip\_file:

        for i, arquivo in enumerate(arquivos):

            novo\_nome = novos\_nomes[i]

            zip\_file.write(arquivo, novo\_nome)  # Adiciona cada arquivo renomeado ao zip

    zip\_buffer.seek(0)  # Volta ao início do buffer para permitir o download

    return zip\_buffer

# Função para gerar novos nomes (exemplo simples)

def gerar\_novo\_nome(pagina\_numero):

    return f"pagina\_renomeada\_{pagina\_numero}.pdf"

# Rota para dividir, renomear e zipar PDFs

@app.route('/api/dividir-renomear-zipar-pdf', methods=['POST'])

@jwt\_required()

def dividir\_renomear\_zipar\_pdf():

    if 'file' not in request.files:

        return jsonify({"message": "Nenhum arquivo enviado"}), 400

    file = request.files['file']

    if file.filename == '':

        return jsonify({"message": "Nenhum arquivo selecionado"}), 400

    if file and file.filename.endswith('.pdf'):

        file\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

        file.save(file\_path)

        # Divide o PDF em várias páginas

        arquivos\_paginas = dividir\_pdf\_em\_paginas(file\_path)

        # Gera novos nomes para cada arquivo

        novos\_nomes = [gerar\_novo\_nome(i + 1) for i in range(len(arquivos\_paginas))]

        # Zipar arquivos renomeados

        zip\_buffer = renomear\_e\_zipar(arquivos\_paginas, novos\_nomes)

        # Remove os arquivos temporários

        for arquivo in arquivos\_paginas:

            os.remove(arquivo)

        os.remove(file\_path)  # Remove o arquivo original

        # Retorna o arquivo zip para download

        return send\_file(zip\_buffer, as\_attachment=True, download\_name='arquivos\_renomeados.zip', mimetype='application/zip')

    return jsonify({"message": "Formato de arquivo inválido"}), 400

# Função para renomear arquivo

def renomear\_arquivo(arquivo\_atual, novo\_nome):

    if os.path.exists(arquivo\_atual):

        os.rename(arquivo\_atual, novo\_nome)

        return True

    return False

# Rota para renomear arquivos

@app.route('/api/renomear-arquivo', methods=['POST'])

@jwt\_required()

def renomear\_arquivo\_route():

    data = request.json

    arquivo\_atual = data.get('arquivo\_atual')

    novo\_nome = data.get('novo\_nome')

    if not arquivo\_atual or not novo\_nome:

        return jsonify({"message": "Parâmetros inválidos"}), 400

    sucesso = renomear\_arquivo(arquivo\_atual, novo\_nome)

    if sucesso:

        return jsonify({"message": "Arquivo renomeado com sucesso!"}), 200

    else:

        return jsonify({"message": "Erro ao renomear arquivo"}), 400

# Rota para dashboard de estatísticas

@app.route('/api/dashboard-stats', methods=['GET'])

@jwt\_required()

def get\_dashboard\_stats():

    total\_holerites = 100  # Exemplo de valor

    erros\_holerites = 5  # Exemplo de valor

    sucesso\_holerites = total\_holerites - erros\_holerites

    stats = {

        "total\_holerites": total\_holerites,

        "sucesso\_holerites": sucesso\_holerites,

        "erros\_holerites": erros\_holerites,

    }

    registrar\_auditoria("Consulta de estatísticas do dashboard", get\_jwt\_identity()['username'], "Sucesso")

    return jsonify(stats), 200

if \_\_name\_\_ == '\_\_main\_\_':

    if not os.path.exists('../logs'):

        os.makedirs('../logs')  # Garante que a pasta de logs exista

    app.run(debug=True, host='0.0.0.0')

# Função para extrair o nome do funcionário de um PDF

def extrair\_nome\_do\_pdf(caminho\_pdf):

    try:

        with pdfplumber.open(caminho\_pdf) as pdf:

            texto = ""

            for pagina in pdf.pages:

                texto += pagina.extract\_text()

            # Procurar a linha que contém "Nome do Funcionário"

            padrao\_nome = re.compile(r'Código\s+Nome do Funcionário\s+CBO\s+Departamento\s+Filial.\*\n(\d+\s+([A-Za-z\s]+))')

            resultado = padrao\_nome.search(texto)

            if resultado:

                nome = resultado.group(2).strip()

                print(f"Nome encontrado: {nome}")  # ADICIONA: Verificar o nome extraído

                return nome

            else:

                print(f"Nome não encontrado no arquivo: {caminho\_pdf}")  # ADICIONA: Verificar se a busca falha

    except Exception as e:

        print(f"Erro ao processar {caminho\_pdf}: {e}")

    return None

# Função para renomear os arquivos com base no nome do funcionário

def renomear\_com\_funcionarios(arquivos\_paginas):

    novos\_nomes = []

    for arquivo in arquivos\_paginas:

        nome\_funcionario = extrair\_nome\_do\_pdf(arquivo)

        if nome\_funcionario:

            novo\_nome = f"{nome\_funcionario}.pdf"

        else:

            novo\_nome = os.path.basename(arquivo)  # Caso não encontre o nome, mantém o nome original

        print(f"Renomeando {arquivo} para {novo\_nome}")  # ADICIONA: Depuração para verificar os novos nomes

        novos\_nomes.append(novo\_nome)

    return novos\_nomes

# Rota para dividir, renomear com nomes de funcionários e zipar PDFs

@app.route('/api/dividir-renomear-zipar-funcionarios', methods=['POST'])

@jwt\_required()

def dividir\_renomear\_zipar\_funcionarios():

    if 'file' not in request.files:

        return jsonify({"message": "Nenhum arquivo PDF enviado"}), 400

    pdf\_file = request.files['file']

    if pdf\_file.filename == '':

        return jsonify({"message": "Nenhum arquivo PDF selecionado"}), 400

    if pdf\_file and pdf\_file.filename.endswith('.pdf'):

        file\_path = os.path.join(UPLOAD\_FOLDER, pdf\_file.filename)

        pdf\_file.save(file\_path)

        # Divide o PDF em várias páginas

        arquivos\_paginas = dividir\_pdf\_em\_paginas(file\_path)

        # Renomeia os arquivos com base no nome dos funcionários

        novos\_nomes = renomear\_com\_funcionarios(arquivos\_paginas)

        # Zipar os arquivos renomeados

        zip\_buffer = renomear\_e\_zipar(arquivos\_paginas, novos\_nomes)

        # Remove os arquivos temporários

        for arquivo in arquivos\_paginas:

            os.remove(arquivo)

        os.remove(file\_path)  # Remove o arquivo PDF original

        # Retorna o arquivo zip para download

        return send\_file(zip\_buffer, as\_attachment=True, download\_name='arquivos\_renomeados\_funcionarios.zip', mimetype='application/zip')

    return jsonify({"message": "Formato de arquivo inválido"}), 400

# Rota de teste para extrair o nome do funcionário de um PDF

@app.route('/api/test-extract', methods=['POST'])

def test\_extraction():

    print("Recebendo a requisição POST")

    if 'file' not in request.files:

        print("Nenhum arquivo foi enviado.")

        return jsonify({"message": "Nenhum arquivo PDF enviado"}), 400

    pdf\_file = request.files['file']

    if pdf\_file.filename == '':

        print("Nenhum arquivo foi selecionado.")

        return jsonify({"message": "Nenhum arquivo PDF selecionado"}), 400

    file\_path = os.path.join(UPLOAD\_FOLDER, pdf\_file.filename)

    pdf\_file.save(file\_path)

    nome\_funcionario = extrair\_nome\_do\_pdf(file\_path)

    if nome\_funcionario:

        print(f"Nome extraído: {nome\_funcionario}")

        return jsonify({"message": f"Nome extraído: {nome\_funcionario}"}), 200

    else:

        print("Nome não encontrado no PDF.")

        return jsonify({"message": "Nome não encontrado"}), 400

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\backend\api\requirements.txt"

blinker==1.8.2

click==8.1.7

colorama==0.4.6

Flask==3.0.3

Flask-Cors==5.0.0

Flask-JWT-Extended==4.6.0

itsdangerous==2.2.0

Jinja2==3.1.4

MarkupSafe==2.1.5

PyJWT==2.9.0

PyPDF2==3.0.1

Werkzeug==3.0.4

C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\node\_modules

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\public\index.html"

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Sistema RH</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\src\components\ConfigPage.tsx"

import React, { useState } from 'react';

import styled from 'styled-components';

const ConfigContainer = styled.div`

  padding: 20px;

`;

const Input = styled.input`

  padding: 8px;

  width: 100%;

  margin-top: 10px;

  margin-bottom: 10px;

  border: 1px solid #ccc;

  border-radius: 4px;

`;

const Button = styled.button`

  padding: 10px 15px;

  background-color: #61dafb;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  &:hover {

    background-color: #21a1f1;

  }

`;

const Message = styled.p<{ success: boolean }>`

  color: ${(props) => (props.success ? 'green' : 'red')};

  font-weight: bold;

`;

const ConfigPage: React.FC = () => {

  const [filePath, setFilePath] = useState('');

  const [message, setMessage] = useState<string | null>(null);

  const [success, setSuccess] = useState<boolean | null>(null);

  const handleSave = () => {

    if (filePath.trim() === '') {

      setMessage('O caminho da planilha não pode estar vazio.');

      setSuccess(false);

    } else {

      setMessage(`Caminho salvo: ${filePath}`);

      setSuccess(true);

    }

  };

  return (

    <ConfigContainer>

      <h2>Configurações</h2>

      <label>Caminho da planilha de funcionários:</label>

      <Input

        type="text"

        value={filePath}

        onChange={(e) => setFilePath(e.target.value)}

        placeholder="Digite o caminho"

      />

      <Button onClick={handleSave}>Salvar Configuração</Button>

      {message && <Message success={success!}>{message}</Message>}

    </ConfigContainer>

  );

};

export default ConfigPage;

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\src\components\Dashboard.tsx"

import React, { useEffect, useState } from 'react';

import styled from 'styled-components';

import { Bar, Pie } from 'react-chartjs-2';

const DashboardContainer = styled.div`

  padding: 20px;

`;

const StatCard = styled.div`

  background-color: #f8f9fa;

  border: 1px solid #ccc;

  padding: 20px;

  border-radius: 4px;

  margin-bottom: 20px;

  text-align: center;

`;

const StatTitle = styled.h3`

  margin-bottom: 10px;

  font-size: 18px;

`;

const StatValue = styled.p`

  font-size: 24px;

  font-weight: bold;

`;

const Dashboard: React.FC = () => {

  const [stats, setStats] = useState({

    total\_holerites: 0,

    sucesso\_holerites: 0,

    erros\_holerites: 0,

  });

  useEffect(() => {

    // Busca os dados do backend

    fetch('http://localhost:5000/api/dashboard-stats', {

      method: 'GET',

      headers: {

        Authorization: `Bearer ${localStorage.getItem('token')}`,

      },

    })

      .then((response) => response.json())

      .then((data) => setStats(data))

      .catch((error) => console.error('Erro ao buscar dados do dashboard:', error));

  }, []);

  const barData = {

    labels: ['Total', 'Sucesso', 'Erros'],

    datasets: [

      {

        label: 'Holerites Enviados',

        data: [stats.total\_holerites, stats.sucesso\_holerites, stats.erros\_holerites],

        backgroundColor: ['#36a2eb', '#4caf50', '#ff6384'],

      },

    ],

  };

  const pieData = {

    labels: ['Sucesso', 'Erros'],

    datasets: [

      {

        data: [stats.sucesso\_holerites, stats.erros\_holerites],

        backgroundColor: ['#4caf50', '#ff6384'],

      },

    ],

  };

  return (

    <DashboardContainer>

      <h2>Dashboard do Sistema</h2>

      <StatCard>

        <StatTitle>Total de Holerites Enviados</StatTitle>

        <StatValue>{stats.total\_holerites}</StatValue>

      </StatCard>

      <StatCard>

        <StatTitle>Sucesso no Envio</StatTitle>

        <StatValue>{stats.sucesso\_holerites}</StatValue>

      </StatCard>

      <StatCard>

        <StatTitle>Erros no Envio</StatTitle>

        <StatValue>{stats.erros\_holerites}</StatValue>

      </StatCard>

      <StatCard>

        <h3>Gráfico de Barras</h3>

        <Bar data={barData} />

      </StatCard>

      <StatCard>

        <h3>Gráfico de Pizza</h3>

        <Pie data={pieData} />

      </StatCard>

    </DashboardContainer>

  );

};

export default Dashboard;

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\src\components\ExtractionPage.tsx"

import React, { useState } from 'react';

const ExtractionPage: React.FC = () => {

  const [file, setFile] = useState<File | null>(null);

  const [downloadUrl, setDownloadUrl] = useState<string | null>(null);

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {

    if (e.target.files && e.target.files[0]) {

      setFile(e.target.files[0]);

    }

  };

  const handleExtractAndRename = () => {

    if (!file) {

      alert('Por favor, selecione um arquivo PDF.');

      return;

    }

    const formData = new FormData();

    formData.append('file', file);

    fetch('http://localhost:5000/api/dividir-renomear-zipar-pdf', {

      method: 'POST',

      headers: {

        Authorization: `Bearer ${localStorage.getItem('token')}`,  // Envia o token JWT no cabeçalho

      },

      body: formData,

    })

      .then((response) => {

        if (response.status === 401) {

          alert('Sessão expirada. Por favor, faça login novamente.');

          return;

        }

        return response.blob();  // Recebe o arquivo zip como blob

      })

      .then((blob) => {

        if (blob) {

          // Cria uma URL temporária para o arquivo zip

          const url = window.URL.createObjectURL(blob);

          setDownloadUrl(url);

        } else {

          console.error('Erro ao processar o arquivo: blob indefinido');

        }

      })

      .catch((error) => console.error('Erro ao processar o arquivo:', error));

  };

  return (

    <div>

      <h2>Dividir, Renomear e Compactar Arquivo PDF</h2>

      <input type="file" accept=".pdf" onChange={handleFileChange} />

      <button onClick={handleExtractAndRename}>Processar PDF</button>

      {downloadUrl && (

        <div>

          <h3>Arquivo Processado:</h3>

          <a href={downloadUrl} download="arquivos\_renomeados.zip">Baixar Arquivo Zipado</a>

        </div>

      )}

    </div>

  );

};

export default ExtractionPage;

"C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\src\components\Header.tsx"

import React from 'react';

import styled from 'styled-components';

import { Link } from 'react-router-dom';

const HeaderContainer = styled.header`

  display: flex;

  justify-content: space-between;

  padding: 20px;

  background-color: #61dafb;

  color: white;

`;

const LogoutButton = styled.button`

  background-color: transparent;

  border: none;

  color: white;

  cursor: pointer;

  font-size: 16px;

  font-weight: bold;

  &:hover {

    text-decoration: underline;

  }

`;

const Header: React.FC = () => {

  return (

    <HeaderContainer>

      <h1>Sistema RH</h1>

      <Link to="/logout">

        <LogoutButton>Sair</LogoutButton>

      </Link>

    </HeaderContainer>

  );

};

export default Header;

sistema-rh\src\components\LoginPage.tsx

import React, { useState } from 'react';

import styled from 'styled-components';

import { useNavigate } from 'react-router-dom';

const LoginContainer = styled.div`

  padding: 20px;

  display: flex;

  flex-direction: column;

  align-items: center;

`;

const Input = styled.input`

  padding: 8px;

  width: 100%;

  margin-top: 10px;

  margin-bottom: 10px;

  border: 1px solid #ccc;

  border-radius: 4px;

`;

const Button = styled.button`

  padding: 10px 15px;

  background-color: #61dafb;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  &:hover {

    background-color: #21a1f1;

  }

`;

const LoginPage: React.FC = () => {

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const [error, setError] = useState('');

  const navigate = useNavigate();  // Para redirecionar após login

  const handleLogin = async () => {

    try {

      const response = await fetch('http://localhost:5000/api/login', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify({ username, password }),

      });

      if (response.status === 401) {

        setError('Usuário ou senha incorretos');

        return;

      }

      const data = await response.json();

      localStorage.setItem('token', data.access\_token);  // Armazena o token JWT no localStorage

      navigate('/');  // Redireciona para a página principal

    } catch (error) {

      console.error('Erro ao fazer login:', error);

      setError('Erro ao fazer login');

    }

  };

  return (

    <LoginContainer>

      <h2>Login</h2>

      <Input

        type="text"

        placeholder="Usuário"

        value={username}

        onChange={(e) => setUsername(e.target.value)}

      />

      <Input

        type="password"

        placeholder="Senha"

        value={password}

        onChange={(e) => setPassword(e.target.value)}

      />

      <Button onClick={handleLogin}>Entrar</Button>

      {error && <p style={{ color: 'red' }}>{error}</p>}

    </LoginContainer>

  );

};

export default LoginPage;

sistema-rh\src\components\Logout.tsx

import React, { useEffect } from 'react';

import { useNavigate } from 'react-router-dom';

const Logout: React.FC = () => {

  const navigate = useNavigate();

  useEffect(() => {

    // Remove o token do localStorage

    localStorage.removeItem('token');

    // Redireciona para a página de login

    navigate('/login');

  }, [navigate]);

  return null;

};

export default Logout;

sistema-rh\src\components\LogsPage.tsx

import React, { useEffect, useState } from 'react';

const LogsPage: React.FC = () => {

  const [logs, setLogs] = useState<string[]>([]);

  useEffect(() => {

    fetch('http://localhost:5000/api/logs', {

      method: 'GET',

      headers: {

        Authorization: `Bearer ${localStorage.getItem('token')}`,

      },

    })

      .then((response) => response.json())

      .then((data) => setLogs(data))

      .catch((error) => console.error('Erro ao buscar logs:', error));

  }, []);

  return (

    <div>

      <h2>Logs de Auditoria</h2>

      {logs.length > 0 ? (

        <ul>

          {logs.map((log, index) => (

            <li key={index}>{log}</li>

          ))}

        </ul>

      ) : (

        <p>Nenhum log disponível</p>

      )}

    </div>

  );

};

export default LogsPage;

sistema-rh\src\components\PrivateRoute.tsx

import React from 'react';

import { Navigate } from 'react-router-dom';

interface PrivateRouteProps {

  children: JSX.Element;

}

const PrivateRoute: React.FC<PrivateRouteProps> = ({ children }) => {

  const token = localStorage.getItem('token');

  if (!token) {

    return <Navigate to="/login" />;

  }

  return children;

};

export default PrivateRoute;

sistema-rh\src\components\RecoverPassword.tsx

import React, { useState } from 'react';

const RecoverPassword: React.FC = () => {

  const [email, setEmail] = useState('');

  const [message, setMessage] = useState('');

  const handleRecover = () => {

    fetch('http://localhost:5000/api/recover-password', {

      method: 'POST',

      headers: {

        'Content-Type': 'application/json',

      },

      body: JSON.stringify({ email }),

    })

      .then((response) => response.json())

      .then((data) => setMessage(data.message))

      .catch((error) => console.error('Erro ao solicitar recuperação de senha:', error));

  };

  return (

    <div>

      <h2>Recuperação de Senha</h2>

      <input

        type="email"

        placeholder="Digite seu e-mail"

        value={email}

        onChange={(e) => setEmail(e.target.value)}

      />

      <button onClick={handleRecover}>Recuperar Senha</button>

      {message && <p>{message}</p>}

    </div>

  );

};

export default RecoverPassword;

sistema-rh\src\components\RenamePage.tsx

import React, { useState } from 'react';

const RenamePage: React.FC = () => {

  const [currentFileName, setCurrentFileName] = useState('');

  const [newFileName, setNewFileName] = useState('');

  const [message, setMessage] = useState<string | null>(null);

  const handleRename = () => {

    if (!currentFileName || !newFileName) {

      alert('Por favor, insira o nome atual e o novo nome do arquivo.');

      return;

    }

    fetch('http://localhost:5000/api/renomear-arquivo', {

      method: 'POST',

      headers: {

        'Content-Type': 'application/json',

      },

      body: JSON.stringify({

        arquivo\_atual: currentFileName,

        novo\_nome: newFileName,

      }),

    })

      .then((response) => response.json())

      .then((data) => setMessage(data.message))

      .catch((error) => console.error('Erro ao renomear arquivo:', error));

  };

  return (

    <div>

      <h2>Renomeação de Arquivos</h2>

      <input

        type="text"

        placeholder="Nome atual do arquivo"

        value={currentFileName}

        onChange={(e) => setCurrentFileName(e.target.value)}

      />

      <input

        type="text"

        placeholder="Novo nome do arquivo"

        value={newFileName}

        onChange={(e) => setNewFileName(e.target.value)}

      />

      <button onClick={handleRename}>Renomear Arquivo</button>

      {message && <p>{message}</p>}

    </div>

  );

};

export default RenamePage;

sistema-rh\src\components\ReportPage.tsx

import React, { useState } from 'react';

import styled from 'styled-components';

import jsPDF from 'jspdf';

import \* as XLSX from 'xlsx';

const ReportContainer = styled.div`

  padding: 20px;

`;

const ButtonGroup = styled.div`

  margin-bottom: 20px;

`;

const Button = styled.button`

  padding: 10px 15px;

  margin-right: 10px;

  background-color: #61dafb;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  &:hover {

    background-color: #21a1f1;

  }

`;

const ErrorTable = styled.table`

  width: 100%;

  border-collapse: collapse;

  margin-top: 20px;

  th, td {

    border: 1px solid #ccc;

    padding: 8px;

    text-align: left;

  }

  th {

    background-color: #f8f9fa;

  }

`;

const Result = styled.div<{ success: boolean }>`

  margin-top: 10px;

  padding: 10px;

  background-color: ${(props) => (props.success ? '#d4edda' : '#f8d7da')};

  border: 1px solid ${(props) => (props.success ? '#c3e6cb' : '#f5c6cb')};

  color: ${(props) => (props.success ? '#155724' : '#721c24')};

`;

interface Report {

  id: number;

  nome: string;

  email: string;

  success: boolean;

  errorReason?: string;

  timestamp?: string;

}

const reportsSimulados: Report[] = [

  { id: 1, nome: 'João Silva', email: 'joao.silva@email.com', success: true },

  { id: 2, nome: 'Maria Souza', email: 'maria.souza@email.com', success: false, errorReason: 'Falha no servidor', timestamp: '2024-09-13 10:35' },

  { id: 3, nome: 'Carlos Lima', email: 'carlos.lima@email.com', success: true },

  { id: 4, nome: 'Ana Costa', email: 'ana.costa@email.com', success: false, errorReason: 'E-mail inválido', timestamp: '2024-09-13 11:00' },

];

const ReportPage: React.FC = () => {

  const [filter, setFilter] = useState<'all' | 'success' | 'error'>('all');

  const filteredReports = reportsSimulados.filter((report) => {

    if (filter === 'success') return report.success;

    if (filter === 'error') return !report.success;

    return true;

  });

  // Função para exportar os relatórios de erro em PDF

  const exportPDF = () => {

    const doc = new jsPDF();

    doc.text('Relatório de Erros no Envio de Holerites', 10, 10);

    let y = 20;

    filteredReports

      .filter((report) => !report.success)

      .forEach((report, index) => {

        doc.text(

          `${index + 1}. ${report.nome} (${report.email}) - Motivo: ${report.errorReason} - Data: ${report.timestamp}`,

          10,

          y

        );

        y += 10;

      });

    doc.save('relatorio-erros-envio.pdf');

  };

  // Função para exportar relatórios em Excel

  const exportExcel = () => {

    const ws = XLSX.utils.json\_to\_sheet(

      filteredReports.map((report) => ({

        Nome: report.nome,

        Email: report.email,

        Sucesso: report.success ? 'Sim' : 'Não',

        'Motivo do Erro': report.errorReason || 'Nenhum',

        'Data/Hora': report.timestamp || 'Não aplicável',

      }))

    );

    const wb = XLSX.utils.book\_new();

    XLSX.utils.book\_append\_sheet(wb, ws, 'Relatórios');

    XLSX.writeFile(wb, 'relatorio-envios.xlsx');

  };

  return (

    <ReportContainer>

      <h2>Relatórios de Envio</h2>

      <ButtonGroup>

        <Button onClick={() => setFilter('all')}>Todos</Button>

        <Button onClick={() => setFilter('success')}>Sucesso</Button>

        <Button onClick={() => setFilter('error')}>Erro</Button>

        <Button onClick={exportPDF}>Exportar PDF</Button>

        <Button onClick={exportExcel}>Exportar Excel</Button>

      </ButtonGroup>

      {filteredReports.length > 0 ? (

        <div>

          {filteredReports.map((report) => (

            <Result key={report.id} success={report.success}>

              {report.nome} ({report.email}):{' '}

              {report.success ? 'Enviado com sucesso' : `Erro: ${report.errorReason}`}

            </Result>

          ))}

          {/\* Tabela de erros \*/}

          {filter === 'error' && (

            <ErrorTable>

              <thead>

                <tr>

                  <th>Nome</th>

                  <th>E-mail</th>

                  <th>Motivo do Erro</th>

                  <th>Data/Hora</th>

                </tr>

              </thead>

              <tbody>

                {filteredReports

                  .filter((report) => !report.success)

                  .map((report) => (

                    <tr key={report.id}>

                      <td>{report.nome}</td>

                      <td>{report.email}</td>

                      <td>{report.errorReason}</td>

                      <td>{report.timestamp}</td>

                    </tr>

                  ))}

              </tbody>

            </ErrorTable>

          )}

        </div>

      ) : (

        <p>Nenhum relatório disponível para esta categoria.</p>

      )}

    </ReportContainer>

  );

};

export default ReportPage;

sistema-rh\src\components\SendPage.tsx

import React, { useState } from 'react';

import styled from 'styled-components';

import jsPDF from 'jspdf';

import \* as XLSX from 'xlsx';

const ReportContainer = styled.div`

  padding: 20px;

`;

const ButtonGroup = styled.div`

  margin-bottom: 20px;

`;

const Button = styled.button`

  padding: 10px 15px;

  margin-right: 10px;

  background-color: #61dafb;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  &:hover {

    background-color: #21a1f1;

  }

`;

const ErrorTable = styled.table`

  width: 100%;

  border-collapse: collapse;

  margin-top: 20px;

  th, td {

    border: 1px solid #ccc;

    padding: 8px;

    text-align: left;

  }

  th {

    background-color: #f8f9fa;

  }

`;

const Result = styled.div<{ success: boolean }>`

  margin-top: 10px;

  padding: 10px;

  background-color: ${(props) => (props.success ? '#d4edda' : '#f8d7da')};

  border: 1px solid ${(props) => (props.success ? '#c3e6cb' : '#f5c6cb')};

  color: ${(props) => (props.success ? '#155724' : '#721c24')};

`;

interface Report {

  id: number;

  nome: string;

  email: string;

  success: boolean;

  errorReason?: string;

  timestamp?: string;

}

const reportsSimulados: Report[] = [

  { id: 1, nome: 'João Silva', email: 'joao.silva@email.com', success: true },

  { id: 2, nome: 'Maria Souza', email: 'maria.souza@email.com', success: false, errorReason: 'Falha no servidor', timestamp: '2024-09-13 10:35' },

  { id: 3, nome: 'Carlos Lima', email: 'carlos.lima@email.com', success: true },

  { id: 4, nome: 'Ana Costa', email: 'ana.costa@email.com', success: false, errorReason: 'E-mail inválido', timestamp: '2024-09-13 11:00' },

];

const ReportPage: React.FC = () => {

  const [filter, setFilter] = useState<'all' | 'success' | 'error'>('all');

  const filteredReports = reportsSimulados.filter((report) => {

    if (filter === 'success') return report.success;

    if (filter === 'error') return !report.success;

    return true;

  });

  // Função para exportar os relatórios de erro em PDF

  const exportPDF = () => {

    const doc = new jsPDF();

    doc.text('Relatório de Erros no Envio de Holerites', 10, 10);

    let y = 20;

    filteredReports

      .filter((report) => !report.success)

      .forEach((report, index) => {

        doc.text(

          `${index + 1}. ${report.nome} (${report.email}) - Motivo: ${report.errorReason} - Data: ${report.timestamp}`,

          10,

          y

        );

        y += 10;

      });

    doc.save('relatorio-erros-envio.pdf');

  };

  // Função para exportar relatórios em Excel

  const exportExcel = () => {

    const ws = XLSX.utils.json\_to\_sheet(

      filteredReports.map((report) => ({

        Nome: report.nome,

        Email: report.email,

        Sucesso: report.success ? 'Sim' : 'Não',

        'Motivo do Erro': report.errorReason || 'Nenhum',

        'Data/Hora': report.timestamp || 'Não aplicável',

      }))

    );

    const wb = XLSX.utils.book\_new();

    XLSX.utils.book\_append\_sheet(wb, ws, 'Relatórios');

    XLSX.writeFile(wb, 'relatorio-envios.xlsx');

  };

  return (

    <ReportContainer>

      <h2>Relatórios de Envio</h2>

      <ButtonGroup>

        <Button onClick={() => setFilter('all')}>Todos</Button>

        <Button onClick={() => setFilter('success')}>Sucesso</Button>

        <Button onClick={() => setFilter('error')}>Erro</Button>

        <Button onClick={exportPDF}>Exportar PDF</Button>

        <Button onClick={exportExcel}>Exportar Excel</Button>

      </ButtonGroup>

      {filteredReports.length > 0 ? (

        <div>

          {filteredReports.map((report) => (

            <Result key={report.id} success={report.success}>

              {report.nome} ({report.email}):{' '}

              {report.success ? 'Enviado com sucesso' : `Erro: ${report.errorReason}`}

            </Result>

          ))}

          {/\* Tabela de erros \*/}

          {filter === 'error' && (

            <ErrorTable>

              <thead>

                <tr>

                  <th>Nome</th>

                  <th>E-mail</th>

                  <th>Motivo do Erro</th>

                  <th>Data/Hora</th>

                </tr>

              </thead>

              <tbody>

                {filteredReports

                  .filter((report) => !report.success)

                  .map((report) => (

                    <tr key={report.id}>

                      <td>{report.nome}</td>

                      <td>{report.email}</td>

                      <td>{report.errorReason}</td>

                      <td>{report.timestamp}</td>

                    </tr>

                  ))}

              </tbody>

            </ErrorTable>

          )}

        </div>

      ) : (

        <p>Nenhum relatório disponível para esta categoria.</p>

      )}

    </ReportContainer>

  );

};

export default ReportPage;

sistema-rh\src\components\SendPage.tsx

import React, { useState } from 'react';

import styled from 'styled-components';

import \* as XLSX from 'xlsx';

const SendContainer = styled.div`

  padding: 20px;

`;

const Button = styled.button`

  padding: 10px 15px;

  background-color: #61dafb;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  &:hover {

    background-color: #21a1f1;

  }

`;

const Input = styled.input`

  padding: 8px;

  width: 100%;

  margin-top: 10px;

  margin-bottom: 10px;

  border: 1px solid #ccc;

  border-radius: 4px;

`;

const FeedbackMessage = styled.p<{ success: boolean }>`

  margin-top: 20px;

  color: ${(props) => (props.success ? '#155724' : '#721c24')};

  background-color: ${(props) => (props.success ? '#d4edda' : '#f8d7da')};

  padding: 10px;

  border: 1px solid ${(props) => (props.success ? '#c3e6cb' : '#f5c6cb')};

`;

const Result = styled.div<{ success: boolean }>`

  margin-top: 20px;

  padding: 10px;

  background-color: ${(props) => (props.success ? '#d4edda' : '#f8d7da')};

  border: 1px solid ${(props) => (props.success ? '#c3e6cb' : '#f5c6cb')};

  color: ${(props) => (props.success ? '#155724' : '#721c24')};

`;

interface Funcionario {

  id: number;

  nome: string;

  email: string;

}

const SendPage: React.FC = () => {

  const [funcionarios, setFuncionarios] = useState<Funcionario[]>([]);

  const [envioStatus, setEnvioStatus] = useState<{ id: number; success: boolean }[]>([]);

  const [feedback, setFeedback] = useState<{ message: string; success: boolean } | null>(null);

  const enviarParaBackend = async (funcionario: Funcionario) => {

    try {

      const response = await fetch('http://localhost:5000/api/send-holerite', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify(funcionario),

      });

      const result = await response.json();

      setFeedback({ message: result.message, success: true });

    } catch (error) {

      setFeedback({ message: 'Erro ao enviar holerite', success: false });

    }

  };

  const handleFileUpload = (e: React.ChangeEvent<HTMLInputElement>) => {

    const file = e.target.files?.[0];

    if (!file) return;

    const reader = new FileReader();

    reader.onload = (event) => {

      const data = new Uint8Array(event.target?.result as ArrayBuffer);

      const workbook = XLSX.read(data, { type: 'array' });

      const sheet = workbook.Sheets[workbook.SheetNames[0]];

      const jsonData = XLSX.utils.sheet\_to\_json<Funcionario>(sheet, { header: 1 });

      const funcionariosData = jsonData.slice(1).map((row: any, index: number) => ({

        id: index + 1,

        nome: row[0],

        email: row[1],

      }));

      setFuncionarios(funcionariosData);

    };

    reader.readAsArrayBuffer(file);

  };

  const handleSend = () => {

    const resultadosEnvio = funcionarios.map((funcionario) => {

      const sucesso = Math.random() > 0.3;

      if (sucesso) {

        enviarParaBackend(funcionario);

      }

      return { id: funcionario.id, success: sucesso };

    });

    setEnvioStatus(resultadosEnvio);

  };

  return (

    <SendContainer>

      <h2>Envio de Holerites</h2>

      <Input type="file" accept=".xlsx" onChange={handleFileUpload} />

      <Button onClick={handleSend} disabled={funcionarios.length === 0}>

        Enviar Holerites

      </Button>

      {feedback && (

        <FeedbackMessage success={feedback.success}>{feedback.message}</FeedbackMessage>

      )}

      {envioStatus.length > 0 && (

        <div>

          <h3>Resultados do Envio:</h3>

          {envioStatus.map((status) => {

            const funcionario = funcionarios.find((f) => f.id === status.id);

            return (

              <Result key={status.id} success={status.success}>

                {funcionario?.nome} ({funcionario?.email}):{' '}

                {status.success ? 'Enviado com sucesso' : 'Erro no envio'}

              </Result>

            );

          })}

        </div>

      )}

    </SendContainer>

  );

};

export default SendPage;

sistema-rh\src\components\Sidebar.tsx

import React from 'react';

import { Link } from 'react-router-dom';

import styled from 'styled-components';

const SidebarContainer = styled.nav`

  width: 200px;

  background-color: #282c34;

  padding: 20px;

  height: 100vh;

  color: white;

`;

const SidebarList = styled.ul`

  list-style-type: none;

  padding: 0;

`;

const SidebarItem = styled.li`

  margin-bottom: 10px;

`;

const SidebarLink = styled(Link)`

  color: white;

  text-decoration: none;

  font-weight: bold;

  &:hover {

    text-decoration: underline;

  }

`;

const Sidebar: React.FC = () => {

  return (

    <SidebarContainer>

      <SidebarList>

        <SidebarItem><SidebarLink to="/configuracoes">Configurações</SidebarLink></SidebarItem>

        <SidebarItem><SidebarLink to="/extracao-arquivos">Extração de Arquivos</SidebarLink></SidebarItem>

        <SidebarItem><SidebarLink to="/envio-holerites">Envio de Holerites</SidebarLink></SidebarItem>

        <SidebarItem><SidebarLink to="/relatorios">Relatórios</SidebarLink></SidebarItem>

        <SidebarItem><SidebarLink to="/">Home</SidebarLink></SidebarItem>

      </SidebarList>

    </SidebarContainer>

  );

};

export default Sidebar;

sistema-rh\src\redux\actions\extractionActions.ts

export const extractFile = (fileName: string) => {

  return {

    type: 'EXTRACT\_FILE',

    payload: `Dados extraídos do arquivo: ${fileName}`,

  };

};

sistema-rh\src\redux\actions\sendActions.ts

export const sendEmails = () => {

  return {

    type: 'SEND\_EMAILS',

  };

};

sistema-rh\src\redux\reducers\extractionReducer.ts

const initialState = {

  extractedData: null,

};

const extractionReducer = (state = initialState, action: any) => {

  switch (action.type) {

    case 'EXTRACT\_FILE':

      return {

        ...state,

        extractedData: action.payload,

      };

    default:

      return state;

  }

};

export default extractionReducer;

sistema-rh\src\redux\reducers\reportReducer.ts

const initialState = {

  ok: ['Relatório 1', 'Relatório 2'],

  errors: ['Erro 1', 'Erro 2'],

};

const reportReducer = (state = initialState, action: any) => {

  switch (action.type) {

    default:

      return state;

  }

};

export default reportReducer;

sistema-rh\src\redux\reducers\sendReducer.ts

const initialState = {

  sent: false,

};

const sendReducer = (state = initialState, action: any) => {

  switch (action.type) {

    case 'SEND\_EMAILS':

      return {

        ...state,

        sent: true,

      };

    default:

      return state;

  }

};

export default sendReducer;

sistema-rh\src\services\logService.ts

export interface LogEntry {

  timestamp: string;

  message: string;

}

const logs: LogEntry[] = [];

export const logEvent = (message: string) => {

  const timestamp = new Date().toISOString();

  logs.push({ timestamp, message });

  console.log(`[${timestamp}] ${message}`);

};

export const getLogs = () => logs;

sistema-rh\src\App.tsx

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import styled from 'styled-components';

import Header from './components/Header';

import Sidebar from './components/Sidebar';

import ConfigPage from './components/ConfigPage';

import ExtractionPage from './components/ExtractionPage';

import SendPage from './components/SendPage';

import ReportPage from './components/ReportPage';

import LoginPage from './components/LoginPage';

import PrivateRoute from './components/PrivateRoute';

import Logout from './components/Logout';

import Dashboard from './components/Dashboard';

const AppContainer = styled.div`

  display: flex;

  height: 100vh;

`;

const ContentContainer = styled.div`

  flex: 1;

  display: flex;

  flex-direction: column;

`;

const MainContent = styled.div`

  flex: 1;

  padding: 20px;

`;

const App: React.FC = () => {

  return (

    <Router>

      <AppContainer>

        <Sidebar />

        <ContentContainer>

          <Header />

          <MainContent>

            <Routes>

              <Route path="/login" element={<LoginPage />} />

              <Route path="/logout" element={<Logout />} />

              <Route path="/configuracoes" element={<PrivateRoute><ConfigPage /></PrivateRoute>} />

              <Route path="/extracao-arquivos" element={<PrivateRoute><ExtractionPage /></PrivateRoute>} />

              <Route path="/envio-holerites" element={<PrivateRoute><SendPage /></PrivateRoute>} />

              <Route path="/relatorios" element={<PrivateRoute><ReportPage /></PrivateRoute>} />

              <Route path="/" element={<h2>Bem-vindo ao Sistema RH</h2>} />

              <Route path="/" element={<PrivateRoute><Dashboard /></PrivateRoute>} /> {/\* Definindo Dashboard como rota padrão \*/}

            </Routes>

          </MainContent>

        </ContentContainer>

      </AppContainer>

    </Router>

  );

};

export default App;

sistema-rh\src\index.tsx

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

const root = ReactDOM.createRoot(

  document.getElementById('root') as HTMLElement

);

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);

C:\Users\rafael.almeida\Desktop\Sistema RH\sistema-rh\src\store.ts

import { configureStore } from '@reduxjs/toolkit';

import sendReducer from './redux/reducers/sendReducer';

import reportReducer from './redux/reducers/reportReducer';

import extractionReducer from './redux/reducers/extractionReducer';

const store = configureStore({

  reducer: {

    send: sendReducer,

    reports: reportReducer,

    extraction: extractionReducer,

  },

});

export default store;

sistema-rh\tsconfig.json

{

  "compilerOptions": {

    "target": "es5",

    "lib": [

      "dom",

      "dom.iterable",

      "esnext"

    ],

    "allowJs": true,

    "skipLibCheck": true,

    "esModuleInterop": true,

    "allowSyntheticDefaultImports": true,

    "strict": true,

    "forceConsistentCasingInFileNames": true,

    "noFallthroughCasesInSwitch": true,

    "module": "esnext",

    "moduleResolution": "node",

    "resolveJsonModule": true,

    "isolatedModules": true,

    "noEmit": true,

    "jsx": "react-jsx"

  },

  "include": [

    "src"

  ]

}