Documentação do Projeto
Rede Social

Criado Por Gustavo Ferreira

Sumário

1-Instalações Necessárias

1.1-Geral

1.1.1-Instalando NodeJS

O nodejs foi a base para a criação de todo o código funcional, seja para importação de bibliotecas e outros itens para dar vida ao site.

Para instalar o NodeJS de forma simples basta ir ao seu navegador (Forma utilizada no ambiente windows) ir ao site:
https://nodejs.org/pt

E então clicar em descarregar e ao fazer isso escolher a biblioteca do seu pc onde o executável será instalado ao terminar o Download vá para o executável e inicie-o e basta clicar em next até o fim e pronto de maneira simples o NodeJs está instalado, para verificar as versões se ok vá para o CMD do seu computador e digite o comando:

nodejs -v

Este comando vai mostrar a versão nodeJs instalada na máquina e também muito necessário digite o comando

npm -version

Este comando vai mostrar a versão do npm instalada ele é totalmente necessário, pois é por ele que fazemos a instalação de bibliotecas do nodejs, caso ocorra algum problema ao verificar a versão instalada é recomendado verificar na pasta PATH do seu Windows se o caminho do nodejs está especificado caso não saiba onde fica a pasta PATH acesse o link a seguir onde nele é explicado:
https://www.autodesk.com/br/support/technical/article/caas/sfdcarticles/sfdcarticles/PTB/Adding-folder-path-to-Windows-PATH-environment-variable.html .

1.1.2-Instalando Visual Studio Code:

Para efetuar o Dowload no visual Studio Code é muitos simples, semelhante ao nodejs vá para o seguinte link

https://code.visualstudio.com

E então clique em Dowload for Windows e então baixar o executável executa-lo e após aceitar os termos recomendo Flegar todas as opções, pois vai facilitar muitos usos posteriormente, após isso bastá prosseguir normalmente com a instalação.

1.2-Front-End

Após finalizar todas as instalações acima vá para uma pasta onde você deseja inserir o projeto e após cria-la clique com o botão direito do mouse e

selecione a opção abrir com o Code (OBS: está opção só estará disponível caso você tenha flegado a mesmo no momento já citado), e então o vscode já irá abrir na pasta selecionada.

1.2.1-Instalação Vite/React

Bom com o VsCode aberto podemos dar inicio a instalação do ambiente react.

No VsCode pressione as teclas CTRL, SHIFT e ‘ essas teclas disparam o atalho para abrir um terminal no vscode, caso você tenha dificuldade dessa forma verifique no topo nele há opções como ARQUIVO, EDITAR, SELEÇÃO entre outros, nessa linha clique em TERMINAL e novo terminal, um adendo ao abrir um terminal verifique onde se encontra o mesmo e veja se não é um terminal powershell no Windows caso seja clique na setinha para baixo ao lado do + e selecione CMD, porque não powershell?

Devido aos critérios de segurança no mesmo ele pode bloquear certas instalações as vezes.

Bom com o terminal aberto e na pasta correta deve ser digitado o seguinte comando :

npm create vite@latest

Primeiramente esse comando irá pedir algumas permissões aperte Y para todas e então em um momento ele vai pedir o Nome para uma pasta de projeto escolha um nome e então prossiga para controlar a navegação em certas partes que contem escolhas utilize as SETAS do Teclado para CIMA e para BAIXO e então selecione no caso desse projeto REACT e após isso Selecione JavaScript e a então o ambiente de trabalho está quase pronto para entrar na nova pasta pelo vscode pode ser utilizado a forma explicada anteriormente ou digite cd NOMEDOPROJETOCRIADO e então digite o seguinte comando:

npm i ou npm install

E aguarde, pois o projeto base está sendo criado, após o termino para fazer um teste utilize o comando:

npm run dev

Ele irá iniciar o serviço web para o front-end.

1.2.2-Instalação de React-Router

Agora instale uma ferramenta que utilizaremos no projeto que é um react-router para criar páginas navegáveis de maneira personalizada na URL, para isso no terminal dentro da pasta do projeto insira o seguinte comando.

npm install react-router react-router-dom

1.2.3-Baixando Extensões Uteis VScode

Agora no VsCode é recomendado para maior praticidade na codificação instalar na aba Extensões localizada no canto esquerdo o seguinte

ES7+ React/Redux/React-Native snippets

Está extensão ajuda na codificação de códigos React uma das principais ajudas é ao criar um novo arquivo .jsx ao utilizar o seguinte comando no arquivo

rfce

Deve aparecer uma sugestão e então ao apertar ENTER ele deixa um corpo react completo para o uso.

1.3-BackEnd

1.3.1-Criação do Ambiente para BackEnd

Para a criação do ambiente para o BackEnd de inicio deve ser criado uma Pasta FORA DO PROJETO FRONT-END, ou seja, ele deve ser inserido na primeira pasta criada para não misturar os ambientes, o estilo deve ficar nessa sistemática.

Pasta do Projeto

|

| _ _ ProjetoDoFront

|_ _ ProjetoDoBackEnd

Ao criar a pasta Abra com o Code para criar os itens da mesma nele é necessário criar uma pasta uploads onde será adicionado itens de mídia enviados pelos usuários e o arquivo servidor.js.

1.3.2-Instalação de Todas as Dependências

Abra um terminal no Code um CMD e rode as seguintes instalações que serão utilizadas:

npm install express express-session cors multer clamdjs bcrypt dotenv mysql2

No trecho acima estão todas as itens a serem instalados para uso no código BackEnd, listarei em tópicos um breve resumo de cada um, pois os mesmos serão explicados no decorrer da sessão 4.

Express – Utilizado no código para criar e gerenciar um servidor backend que recebe requisições do Front-End via APIs REST, processa os dados e retorna respostas.

Express-session: Utilizado para gerenciar sessões no backend, permitindo armazenar e recuperar informações do usuário entre requisições. Ele cria um identificador de sessão no navegador via cookie e mantém os dados da sessão no servidor, podendo ser usado para verificar autenticação e tratar permissões em APIs REST.

Cors: Middleware que permite ao servidor definir quais origens (domínios) podem acessar seus recursos, ou seja, o front-end criado, evitando bloqueios pela política de mesma origem. Ele é usado para autorizar requisições entre diferentes domínios, sendo essencial em APIs REST acessadas por um front-end hospedado em outro domínio.

Multer: Utilizado para processar uploads de arquivos recebidos via requisições HTTP. Ele permite armazenar os arquivos no servidor, seja em uma pasta especificada ou na memória, facilitando o gerenciamento de uploads de áudios, vídeos, imagens e documentos.

Clamdjs: Biblioteca que se comunica com o antivírus ClamAV (ClamD) presente na máquina local ou em um servidor remoto. Ele permite enviar arquivos para verificação e detectar conteúdos potencialmente maliciosos antes de serem processados ou armazenados no servidor, ajudando a prevenir a propagação de malware.

Bcrypt: Utilizado no código para criação de Hash de senha digitados no Cadastro Válido de Usuarios, e utiliza números de salts para aumentar a dificuldade de ataques.

Dotenv: Utilizado para uso de Variáveis de ambiente no código podendo em um arquivo .env guardar dados importantes e que não devem ser expostos.

Mysql2:Utilizado para fazer as requisições ao banco de dados onde nele são feito chamadas sql para retornar, inserir ou atualizar valores.

1.4-Banco de Dados

1.4.1-Instalando o Xampp

O xampp é um app criado para fazer a instalação e uso de um servidor Apache, mas também junto dele contém um banco de dados MySql com uma ótima interface web e de fácil uso para criação das tabelas necessárias e armazenamento dos dados.

Para fazer a instalação deve-se ir ao site:

https://www.apachefriends.org/pt_br/index.html

Nele contém os dowloads para múltiplos dispositivos basta instalar o do seu sistema operacional e utilizado em caso de Windows, basta executar o installer e seguir até o final com a instalação.

1.4.1.1- Iniciando Banco de Dados

Quando a instalação for finalizada vá para o app criado podendo estar na área de trabalho ou então pesquisando na barra do Windows, ao abrir o app clique em start para o server APACHE, pois nele está a interface personalizada e start MySql, assim iniciando o banco de dados, para ir direto ao site com a interface clique na barra do Mysql em Admin e o mesmo irá abrir o site automaticamente, os detalhes de tabelas criadas e códigos serão explicados na sessão 5.

1.5-Instalando Anti-Virus do servidor

Para tratar o recebimento de arquivos maliciosos na página foi utilizado o anti-virus local clamd, que contém suporte para plataforma Windows.

1.5.1-Instalação do Clamd

A instalação do Clamd é um pouco complexa, porém tentarei explicar da melhor forma para o entendimento, para baixar o executável entre no link a seguir e selecione a sua plataforma.

https://www.clamav.net/downloads

Recomendo para pessoas sem muito conhecimento que seja instalado a versão clamav-1.4.2.win.x64.msi, porque a mesma por ser x64 é uma versão para pcs de 64 bits e com suporte aos processadores INTEL e AMD a versão em questão pode ser instalado uma superior, porém no dia da criação desse documento a versão 1.4.2 é a mais atual.

Ao clicar para efetuar a instalação é possível que o mesmo seja bloqueado pelo Windows em uma tela azul clique o item Mais Informações ou Avançado e então clique em INSTALAR MESMO ASSIM e prossiga com a instalação, um adendo atente-se no local de instalação dos arquivos, pois entraremos no diretório do app no fim da intalação.

1.5.2-Configuração do Clamd

Finalizando a instalação agora será necessário efetuar configurações para seu funcionamento, vá para a pasta onde ele foi instalado por padrão do Windows fica em C:\Program Files\ClamAV e então será necessário copiar 2 arquivos e editar os mesmo, atente-se para não apaga-los, via comando para fazer a cópia será:

copy .\conf_examples\freshclam.conf.sample .\freshclam.conf

copy .\conf_examples\clamd.conf.sample .\clamd.conf

Ou de maneira padrão entre na pasta conf_examples e copie os arquivos freshclam.conf.sample e clamd.conf.sample e então os renomeie para freshclam.conf e clamd.conf com os nomes modificados os insiram no diretório anterior, ou seja, no C:\Program Files\ClamAV e então abra esses arquivos com o qualquer editor de texto, de preferencia o Visual Studio Code e apenas remova o item Example sem um # no inicio e pronto, faça isso nos 2 arquivos.

Agora para iniciar o Clamd procure na pasta o freshclam.exe e execute o mesmo como admin, ele atualiza o banco de dados de itens maliciosos que sempre são atualizados e então quando finalizar feche se necessário e execute o clamd.exe e aguarde, este console deve ficar ligado para o anti-virus ficar ativado no seu console deve ficar da seguinte forma.

Limits: Global time limit set to 120000 milliseconds.

Limits: Global size limit set to 419430400 bytes.

Limits: File size limit set to 104857600 bytes.

Limits: Recursion level limit set to 17.

Limits: Files limit set to 10000.

Limits: MaxEmbeddedPE limit set to 41943040 bytes.

Limits: MaxHTMLNormalize limit set to 41943040 bytes.

Limits: MaxHTMLNoTags limit set to 8388608 bytes.

Limits: MaxScriptNormalize limit set to 20971520 bytes.

Limits: MaxZipTypeRcg limit set to 1048576 bytes.

Limits: MaxPartitions limit set to 50.

Limits: MaxIconsPE limit set to 100.

Limits: MaxRecHWP3 limit set to 16.

Limits: PCREMatchLimit limit set to 100000.

Limits: PCRERecMatchLimit limit set to 2000.

Limits: PCREMaxFileSize limit set to 104857600.

Archive support enabled.

Image (graphics) scanning support enabled.

Detection using image fuzzy hash enabled.

AlertExceedsMax heuristic detection disabled.

Heuristic alerts enabled.

Portable Executable support enabled.

ELF support enabled.

Mail files support enabled.

OLE2 support enabled.

PDF support enabled.

SWF support enabled.

HTML support enabled.

XMLDOCS support enabled.

HWP3 support enabled.

OneNote support enabled.

Self checking every 600 seconds.

E então agora seu anti-virus clamd está configurado e funcional.

2-Organização de Pastas e Arquivos

Dentro da pasta no projeto Backend e Front-End é necessário a criação de um arquivo .env para uso das variáveis de ambiente com informações que não devem ser visíveis ao usuário então basta entrar no projeto e criar esses arquivos, dentro deles serão armazenados.

BackEnd:

-Host do Banco de dados

-Usuário do Banco de dados

-Senha do Usuário do Banco de dados

-Banco a ser utilizado

-Chave especial para criação da sessão de usuário

-Chave Secreta para Criptografia e Descriptografia de Dados.

Agora no Front-End nele foi armazenado:

-Link para envio de requisições ao Back-End

Dentro do ambiente Vite criado na pasta src crie as pastas onde serão armazenadas as páginas do seu app, no meu caso eu fiz a criação da pasta pages e crie outra pasta para inserção de componentes reutilizáveis por exemplo o Footer e o NavBar da página, eu fiz o uso da pasta assets para carregamento de imagens, porém é recomendado inseri-los na pasta public, pois isso me rendeu problemas de carregamento das imagens no deploy do site Versel.

As imagens forma criadas utilizando IA ou o Bootstrap icons na web o mesmo pode ser encontrado caso seja necessário.

3-Código e Arquivos Front-End Explicado

Agora nessa sessão irei explicar as funções utilizadas imports e tudo mais de cada arquivo do Front-End, porém devo salientar que vou ressaltar lógicas e usos do react, estilizações CSS e uso de Tags sem uso em funções não serão explicados.

3.1-Estrutura de Código React

Por padrão a estrutura de códigos react utilizam certas semelhanças com o JavaScript e Html onde nele se utilizam funções JS e Tags para renderização por exemplo, para estruturarmos o inicio de uma página fazemos da seguinte maneira.

Imports do código <-Bibliotecas, componentes de páginas, css de páginas devem ser inseridos dentro dos imports, por exemplo,

import './Pagina1.css'; <-Esse trecho importa o css de Pagina1 no diretório atual.

function Página1() <-função para carregamento dos componentes da página essa de preferencia deve ter o nome do arquivo, por exemplo, deve estar dentro do arquivo Pagina1.jsx e também uso de letras maiúsculas de inicio evita problema no carregamento dos componentes no momento de importação para outro código.

{

Neste trecho antes do return podemos adicionar lógicas para manipulação personalizadas dos componentes do return , por exemplo:

let usuario = “Gustavo”

return <-Componentes personalizados a serem retornados(

< > <-Abertura de uma tag pai onde dentro dele ficaram todas as tags a serem utilizadas

{usuario} <-Tag a ser Renderizada e {usuario} variável a ser carregada neste exemplo retorna “Gustavo”, ou seja, na página irá carregar um parágrafo com valor Gustavo

</> <-Fechamento da tag pai

)

}

Export default Pagina1 <-Neste trecho exporta a função para a mesma ser carregada no momento que houver o import da mesma em outros códigos.

3.2-Arquivo main.jsx

Em react o arquivo main assim como o app.jsx vem por padrão, mas o que ele faz?

Bom ele é o arquivo que faz o carregamento dos componentes no html onde os mesmos estarão visíveis, vou fazer a explicação com base no exemplo abaixo:

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'
createRoot(document.getElementById('root')).render(
 <StrictMode>
  <App />
 </StrictMode>,
)
```

Neste código ele importa o strict usado para verificações detalhadas que aparecem no console da página como erros, ou warnings, no caso ele pode ser retirado posteriormente quando o código for utilizado em produção, ele também importa o createRoot ele é o que faz a integração do código em uma tag especifica no html, fazendo assim toda a renderização e ele também importa o app que será explicado na próxima sessão seu uso.

Primeiro no trecho

```
createRoot(document.getElementById('root')).render(
```

Ele está basicamente pegando uma tag de id”root” em um html e então como podemos verificar abaixo:

```
<StrictMode> <App /> </StrictMode>
```

Ele está renderizando nesse tag App em forma de Debug com o Strict, ou seja, será carregado os elementos de App e irá aparecer no console problemas, caso tenha, e possíveis causas.

Mas como funciona no HTML?

Simples na pasta principal se você analisar por padrão tem um html de nome index e ao analisar o código dele podemos ver que dentro da tag body contém uma div de id=”root” ou seja nessa tag serão adicionado os componentes do app e também e MUITO IMPORTANTE!

Para o carregamento a tag:

<script type="module" src="/src/main.jsx"></script>

É de suma importância, pois nele que é importado o código main.jsx para uso no html, ou seja, ele que faz o id root funcionar na página.

3.3-Arquivo App.jsx

O arquivo App.jsx é um dos principais arquivos para importação das páginas criadas nele foram desenvolvidos os ROUTERS para páginas, segue e código e a explicação a seguir:

```
import {BrowserRouter, Routes, Route, Navigate} from 'react-router-dom'

import Home from './pages/Home/Home'

import Chat from './pages/Chat/Chat'

import Footer from './component/Footer'

import Login from './pages/TelaLogin/Login'

import PerfildeUsuario from './pages/PerfildeUsuario/PerfildeUsuario'

import OutroUser from './pages/AbrirOutroPerfilUser/OutroUser'

import Publicacao from './pages/Publicacoes/Publicacao'

function App() {

 return (

   <>

   <BrowserRouter>

    <Routes>

      <Route path="/" element={<Home />}/>

      <Route path="/Chat" element={<Chat />} />

      <Route path='/Login' element={<Login/>}/>

      <Route path='/PerfilDeUsuario' element={<PerfildeUsuario/>}/>

      <Route path='/VerPerfil' element={<OutroUser/>}/>
```

```
      <Route path='/Publicacoes' element={<Publicacao/>}/>
      </Routes>
    </BrowserRouter>
    <Footer/>
  </>
 )
}
export default App
```

import {BrowserRouter, Routes, Route, Navigate} from 'react-router-dom'

Utilizando essa importação torna possível criar Urls personalizadas para criação das páginas desenvolvidas, onde ele foi utilizado?

```
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<Home />}/>
      <Route path="/Chat" element={<Chat />} />
      <Route path='/Login' element={<Login/>}/>
      <Route path='/PerfilDeUsuario' element={<PerfildeUsuario/>}/>
      <Route path='/VerPerfil' element={<OutroUser/>}/>
      <Route path='/Publicacoes' element={<Publicacao/>}/>
      </Routes>
    </BrowserRouter>
```

O browser Routes é uma tag onde ao ser trocado a url ele verifique se a rota acessada está presente na tag Routes, que por sua vez armazena todas as rotas presentes e a tag Route representa a rota em si, criada no path=”OriginDaUrl” no trecho path o valor colocado tem que ser no mínimo uma / nele fica armazenado a rota e no caso do / sozinho essa seria a primeira página a ser carregada por padrão e dentro de element colocamos o código .jsx que contém a página especificada no meu exemplo eu utilizei element={<Home />}/> para esse componente funcionar vamos explicar por etapas para melhor entendimento.

Para o funcionamento da tag <Home/>, primeiro nos imports verifique o trecho:

```
import Home from './pages/Home/Home'
```

Neste trecho ele importa a Função exportada de nome Home presente no Diretório ./pages/Home/Home.jsx <- na pasta ele procura automaticamente o código jsx por isso o mesmo é chamado sem extenção e então ao utilizar a Tag <Home /> que faz referência a importação de Home o mesmo ao ser digitado a / na url da página ele será direcionado a Home onde nele contém o código da página Home. E nesse caso todas as rotas tem o mesmo comportamento, já o <Footer/> ele é um componente reutilizável então ao deixarmos no APP todas as páginas vão carregar esse componente automaticamente.

E por fim exportamos essa tag para uso em outro código que neste caso será utilizado na main.

3.4-Arquivos página Home

Em Home fiz apenas o uso do Navbar presente na pasta de componentes sem uma lógica especifica, porém no Navbar importado presente na página componentes há uma lógica que deve ser explicada:

Em Navbar há as seguintes funções:

```
const apiUrl = import.meta.env.VITE_BackEndUrl;


async function LogarPagina() {
  const response = await fetch(` ${apiUrl}/protected` , {
    method: 'GET',
    credentials: 'include', // Inclui cookies na requisição
  });
  const result = await response.json();
  if (result.status != "200") {
    document.getElementById('LinkPerfil').className = 'd-none'
    return ('NãoLogado')
  }
  else {
    document.getElementById('Logar/Logout').innerText = 'Efetuar Logout'
```

```
    document.getElementById('LinkPerfil').className = 'btn btn-outline-light ml-4 text-light'
    return ('Logado')
  }
 }
 LogarPagina();

 async function Logout() {
  const response = await fetch(`${apiUrl}/logout`, {
   method: 'POST',
   headers: {
    'Content-Type': 'application/json',
   },
   credentials: 'include', // Inclui cookies na requisição
  })
  const result = await response.json();
  console.log(result.message)
  document.getElementById('Alerta').innerHTML = '<div id="AlertaLogout" class="alert alert-success alert-dismissible fade show text-center" role="alert"><strong>Logout Efetuado com Sucesso </strong>' + result.message + '<button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>';
  document.getElementById('Logar/Logout').innerText = 'Logar'
  document.getElementById('LinkPerfil').className = 'd-none'
 }
```

Bom vamos por partes
Primeiro:

```
const apiUrl = import.meta.env.VITE_BackEndUrl;
```

criamos uma const de nome apiUrl e seu valor é impor.meta.env.VITE_BACKEndUrl este import não está fazendo nada mais do que importando a variável de ambiente que faz referencia ao meu servidor BackEnd,

ou seja, VITE_BACKEndUrl é igual a “http://localhost:3000/” , pois esse é o valor criado.

Agora a Função LogarPagina:

```
async function LogarPagina() {
  const response = await fetch(` ${apiUrl}/protected` , {
    method: 'GET',
    credentials: 'include', // Inclui cookies na requisição
  });
  const result = await response.json();
  if (result.status != "200") {
    document.getElementById('LinkPerfil').className = 'd-none'
    return ('NãoLogado')
  }
  else {
    document.getElementById('Logar/Logout').innerText = 'Efetuar Logout'
    document.getElementById('LinkPerfil').className = 'btn btn-outline-light ml-4 text-light'
    return ('Logado')
  }
 }
```

Ela envia de início uma requisição utilizando método GET para ${apiUrl}/protected, MUITO IMPORTANTE na requisição é necessário incluir credentials, pois nela está incluída o cookie de sessão armazenado, o retorno esperado dessa função é um status 200 para caso usuário esteja logado ou Diferente de 200 caso deslogado e dependendo do caso. Caso ele esteja logado ele faz o seguinte:

```
    document.getElementById('Logar/Logout').innerText = 'Efetuar Logout'
    document.getElementById('LinkPerfil').className = 'btn btn-outline-light ml-4 text-light'
    return ('Logado')
```

Primeiro ele pega o valor da Tag com ID 'Logar/Logout' e modifica seu texto para Efetuar Logout e depois ele paga a Tag com ID LinkPerfil e modifica seu className para 'btn btn-outline-light ml-4 text-light', este são itens personalizados usando Bootstrap e retorna (“Logado”) para uso posterior.

Agora caso o usuário não esteja logado:

```
document.getElementById('LinkPerfil').className = 'd-none'

return ('NãoLogado')
```

Ele apenas define a class da tag de Id LinkPerfil como “d-none” e retorna NãoLogado para uso posterior.

Após essa função ela é chamada por meio do

LogarPagina()

No fim qual o uso dessa função? Caso o usuário já esteja logado ele modifica os botões de forma personalizada para efetuar logout e libera o uso do boão LinkPerfil que direciona o usuário para seu perfil.

Essa função também é chamada para verificação ao usuário clicar no botão perfil e Efetuar_Logout ou Login utilizando seus returns para encaminhar o usuário caso Logado e Caso NãoLogado. A seguir os trechos de código onde isso é representado.

```
<li className="nav-item" id="item-hover">
 <a id="LinkPerfil" onClick={() => {
  const Valor = LogarPagina()
   .then((funcional) => {
    if (funcional == 'Logado') {
     window.location.href = '/PerfilDeUsuario'
    }
    else
    {
     window.location.href = '/Login'
    }
   })
 }}  className='d-none' >Perfil de Usuário</a>
```

```
          </li>
          <li className="nav-item">
           <a onClick={() => {
             const Valor = LogarPagina()
              .then((funcional) => {
               if (funcional == 'Logado') {
                Logout();
               }
               else
               {
                window.location.href = '/Login'
               }
              })
           }} id="Logar/Logout" className="btn btn-outline-light ml-4 text-
light">Logar</a>
        </li>
```

Agora a Função Logout():

```
 async function Logout() {
  const response = await fetch(` ${apiUrl}/logout` , {
   method: 'POST',
   headers: {
    'Content-Type': 'application/json',
   },
   credentials: 'include', // Inclui cookies na requisição
  })
  const result = await response.json();
  console.log(result.message)
  document.getElementById('Alerta').innerHTML = '<div id="AlertaLogout"
class="alert alert-success alert-dismissible fade show text-center"
```

```
role="alert"><strong>Logout Efetuado com Sucesso </strong>' + result.message +
'<button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button></div>';

  document.getElementById('Logar/Logout').innerText = 'Logar'

  document.getElementById('LinkPerfil').className = 'd-none'

 }
```

Seu nome meio que já se auto explica caso o usuário estiver logado em uma seção válida ele o desloga fazendo ele ter que se logar novamente para isso ele envia uma requisição utilizando método POST para ${apiUrl}/logout e então modifica a tag de id Alerta para parecer e modifica o texto de Logar/Logout para ‘Logar’ e modifica o classname da tag LinkPerfil para d-none.

3.5-Arquivos página TelaLogin

Devido ao mesmo ter uma quantidade maior de funções vou dividi-las e explica-las:

Primeiro ele importa o NavbarLogin, pois sua navbar contém elementos diferentes da home, e então cria uma constante com a Váriavel de ambiente relatada no código Home.

E então agora temos a primeira função a Login():

```
const apiUrl = import.meta.env.VITE_BackEndUrl;

function Login() {

 async function LoginTela() {

  const response = await fetch(` ${apiUrl}/protected` , {

   method: 'GET',

   credentials: 'include', // Inclui cookies na requisição

  });

  const result = await response.json();

  if (result.status == "400") {

  }

  else {

   window.location.replace("/PerfilDeUsuario")

   return
```

```
  }
 }
 LoginTela();
```

Essa função é semelhante a anterior citada em home, porém seu uso é único nesse caso, ela verifica se o usuário está ou não autenticado caso esteja ele redireciona o usuário para sua tela de perfil assim evitando o acesso indevido.

Agora a função Logar():

```
 async function Logar() {
  let Email = document.getElementById('InputEmail').value;
  let Senha = document.getElementById('InputSenha').value;
  const response = await fetch(`${apiUrl}/login`, {
   method: 'POST',
   headers: {
    'Content-Type': 'application/json' },
   credentials: 'include', // Inclui cookies na requisição
   body: JSON.stringify({ email: Email, senha: Senha }),
  });
  const result = await response.json();
  if (response.status == "400") {
   document.getElementById('Alerta').innerHTML = '<div id="AlertaErro"
class="alert alert-danger alert-dismissible fade show"
role="alert"><strong>Erro  </strong>' + result.message + '<button type="button"
class="close" data-dismiss="alert" aria-label="Close"><span aria-
hidden="true">&times;</span></button></div>';
  }
  else {
   document.getElementById('Alerta').innerHTML = '<div id="AlertaErro"
class="alert alert-success alert-dismissible fade show"
role="alert"><strong>Usuário Vinculado com Sucesso </strong>' + result.message
+ '<button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button></div>';
```

```
    document.getElementById('InputEmail').value = "";
    document.getElementById('InputSenha').value = "";
    setTimeout(() => window.location.replace("/PerfilDeUsuario"), 1000)
  } }
```

Essa função faz exatamente o que o nome retrata ela faz o envio dos dados digitados pelo usuário nos inputs de if “InputEmail” e “InputSenha” enviando para ser tratado no BackEnd, via requisição POST com o body com os dados e o retorno esperado é status 400 em caso de Erro e 200 caso o mesmo seja encontrado no banco de dados as tratativas são o seguinte em caso de erro ele personaliza a tag de id “Alerta” explicando o Erro e caso, seja sucesso ele Modifica essa tag Alerta para sucesso e após 1 segundo envia o usuário para /PerfilDeUsuario como pode se ver em setTimeout()

Função Criar Usuário ():

```
async function CriarUsuario() {
  let Nome = document.getElementById('NomeUsuario').value;
  let Email = document.getElementById('Email').value;
  let Senha = document.getElementById('SenhaCadastro').value;
  let nickname = document.getElementById('Nick').value;
  console.log(Nome)
  const response = await fetch(` ${apiUrl}/usuarios` , {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    credentials: 'include', // Inclui cookies na requisição
    body: JSON.stringify({ nome: Nome, email: Email, senha: Senha, nick:
nickname }),
  });
  const result = await response.json();
  if (response.status != "200") {
```

```
    document.getElementById('Alerta').innerHTML = '<div id="AlertaErro"
class="alert alert-danger alert-dismissible fade show"
role="alert"><strong>Erro  </strong>' + result.message + '<button type="button"
class="close" data-dismiss="alert" aria-label="Close"><span aria-
hidden="true">&times;</span></button></div>'; }

  else {

    document.getElementById('Alerta').innerHTML = '<div id="AlertaErro"
class="alert alert-success alert-dismissible fade show"
role="alert"><strong>Usuário Criado com Sucesso  </strong>' + result.message +
'<button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button></div>';

    document.getElementById('NomeUsuario').value = "";

    document.getElementById('Email').value = "";

    document.getElementById('SenhaCadastro').value = "";

    document.getElementById('Nick').value = "";

    TrocaCadastro();

  } }
```

Essa função é disparada ao usuário clicar no botão para Cadastrar um novo usuário, ela pega as informações nos inputs digitadas pelo usuário e então as envia via POST para o BackEnd e caso o status de resposta seja diferente de 200, então ele insere um alerta de erro personalizado e caso seja igual a 200 ele insere um alerta de sucesso limpa os campos e roda a função TrocaCadastro() que será explicada posteriormente.

Funções TrocaView() e TrocaView2:

```
 function TrocaView() {

  let img = document.getElementById('SenhaVisivel').src

  let inputsenha = document.getElementById('InputSenha').type

  if (inputsenha == 'password') {

    document.getElementById('SenhaVisivel').src = './src/assets/MostrarSenha.png'

    document.getElementById('InputSenha').type = 'text'  }

  else {
```

```
    document.getElementById('SenhaVisivel').src =
'./src/assets/EsconderSenha.png'
    document.getElementById('InputSenha').type = 'password'
  }
 }
 function TrocaView2() {
  let img = document.getElementById('SenhaVisivel2').src
  let inputsenha = document.getElementById('SenhaCadastro').type
  if (inputsenha == 'password') {
    document.getElementById('SenhaVisivel2').src =
'./src/assets/MostrarSenha.png'
    document.getElementById('SenhaCadastro').type = 'text'
  }
  else {
    document.getElementById('SenhaVisivel2').src =
'./src/assets/EsconderSenha.png'
    document.getElementById('SenhaCadastro').type = 'password'
  }
 }
```

Essas funções acima são simplesmente para criar um botão personalizado para Aparecer e esconder a senha sendo TrocaView() para tela de login e TrocaView2() para tela de cadastro ambas verificando o type do input e caso seja igual password ele o troca para Text e assim vice-versa.

Função TrocaCadastro():

```
 function TrocaCadastro() {
  let Tela = document.getElementById('FundoCadastro').className;
  let Tela2 = document.getElementById('FundoLogin').className;
  document.getElementById('Colunas').style.animationPlayState = 'running';
  document.getElementById('Colunas2').style.animationPlayState = 'running';
  document.getElementById('FundoCadastro').className = Tela2;
```

```
  document.getElementById('FundoLogin').className = Tela;
 }
```

Essa função também tem uso simples para fazer uma transição de tela personalizada ao ser clicado no botão para alterar entre Cadastro e Login e por ultimo.

Função setTimeout solta:

```
 let janela = window.location.href;
 janela = janela.replace("http://localhost:5173/Login","");
setTimeout(()=>{
 if (janela != "") {
 document.getElementById('Alerta').innerHTML = '<div id="AlertaErro" class="alert alert-danger alert-dismissible fade show" role="alert"><strong>Erro  </strong> Necessario Efetuar o Login Primeiro <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>';
}
}
,500)
```

Essa função verifica se no cabeçalho da página não contém nenhuma requisição Get de erro que se dá caso tenha tido entrada não Autorizada em Páginas protegidas caso tenha ele imprime um alerta de erro personalizado.

3.6-Arquivos página PerfildeUsuario

Bom vamos agora a uma parte bem extensa, pois contém muitas funcionalidades vamos por partes:

Primeiro fazemos o import da NavBarPerfil para ser a Navbar da Página e também fizemos uma importação inédita o:

```
import React, { useEffect, useState } from "react";
```

Esse import adiciona uma funcionalidade que o react tem que é para carregamentos em tempo real de itens onde caso um valor seja atualizado esses itens podem ser atualizados assincronamente dando uma melhor experiência para o usuário.

Agora utilizamos a const citada em sessões anteriores a apiUrl para uso no código e então declaramos os seguintes usestates:

```
const [dadosUser, setDadosUser] = useState([]);

const [CarregaFeed, setCarregaFeed] = useState([])
```

Bom a primeira dadosUser serão usados para carregamento de dados de usuário no perfil, como nome, bio, foto de perfil e número de seguidores.

Já o CarregaFeed será utilizado para carregar a parte de publicações do usuário, os Posts e Comentários .

Função CarregaFeed:

```
const CarregaoFeed = async () => {
 try {
  const response = await fetch(` ${apiUrl}/protected/EnviaPubliFront` , {
   method: 'get',
   credentials: 'include',
  });
  const result = await response.json();
  if (result.status === '400') {
  } else {
   setCarregaFeed(result.message);
  }
 } catch (error) {
  console.error('Erro ao carregar o perfil:', error);
 }
}
```

Nessa função é feito um try que envia uma requisição Get ao BackEnd e os valores retornados são os dados que compõem a tela de usuário, os mesmo são armazenados em setCarregaFeed(result.message);

E então após essa função temos a função carregaPerfil:

```
const carregaPerfil = async () => {
 try {
  const response = await fetch(` ${apiUrl}/protected` , {
```

```
      method: 'GET',
      credentials: 'include',
    });
    const result = await response.json();
    if (result.status === '400') {
      window.location.replace('/Login?erro');
    } else {
      setDadosUser(result.dados);
    }
  } catch (error) {
    console.error('Erro ao carregar o perfil:', error);
  } finally {
  }
};
```

Ela faz o mesmo que a anterior, porém com duas diferenças em caso de erro 400 retornado pelo servidor ele retorna o usuário para tela de login, pois esse erro representa que o usuário não está logado e também ele guarda os dados recebidos para serem utilizados em setDadosUser(result.dados).

Em conjunto temos o próximo código que faz o carregamento a cada 5 segundos desses dados para verificação em caso de novas informações:

```
useEffect(() => {
  let isMounted = true;
  const fetchProfile = async () => {
    if (isMounted) {
      await carregaPerfil();
      await CarregaoFeed();
      setTimeout(fetchProfile, 5000);
    }
  };
```

```
    fetchProfile();
    return () => {
      isMounted = false;
    };
  }, []);
```

Este trecho ele chama as funções criadas acima.

Agora para darmos contexto no uso de CarregaFeed() e carregaPerfil(), vamos analisar os itens que são retornados:

Em CarregaFeed recebemos um objeto com os seguintes itens

Comentario_Author

Conteudo_Publicacao

NomeExtesao

comentarios: Array(1)

1. **0**:

    comentario:

    nickname:

Data

id

total_comentarios: 0

Bom com esses valores temos o comentário do Autor, ou seja, se houver um comentário do autor ele será utilizado junto com o vídeo, Conteudo_Publicacao, nele contém o link para o vídeo em especifico armazenado no diretório upload, NomeExtensao apenas o nome do arquivo para manipulação do tipo de arquivo caso seja imagens ou vídeos, comentários um array onde nele temos o nickname da pessoa que comentou na publicação e o comentário do comentador a Data da publicação e o id da publicação utilizado para manipular dados lógicos em caso seja apagado curtido entre outros, esses dados são tratados e utilizados em uma função map() no corpo da aplicação para a exibição de conteúdos personalizados a função map é semelhante ao forEach.

Agora em carregaPerfil são vinculados os seguintes dados:

email

foto_perfil

id

link_foto

mensagem_bio

nickname

nome

total_seguidores

total_seguindo

Esses dados são utilizados para personalizar os dados de usuário então foto_perfil contem a foto atual do usuário, email contém o email o id para uso em certas lógicas, link_foto representa o link para carregamento da foto, mensagem_bio a bio atual do usuário, nickname representa o nick atual, nome nele contém o nome do usuário, total_seguidores total_seguindo contém a quantidade numérica de seguidores do usuário

Como em setDadosUser esta sendo usado dados onde contém esses itens, para uso basta inserir no front-end {dadosUser.nickname} por exemplo o valor retornado dentro de {} é o nickname de usuário e assim podendo adicionar e modificar esses dados em tempo real.

Agora a função para alterar a foto de perfil, EnviaFotoPerfil:

```
function EnviarFotoPerfil() {
  const fileInput = document.getElementById('EnviaFotoPerfil');
  fileInput.click();
  fileInput.addEventListener('change', async () => {
    const file = fileInput.files[0];
    if (file) {
      const formData = new FormData();
      formData.append('image', file);
      try {
        const response = await fetch(`${apiUrl}/protected/envio/fotoPerfil`, {
          method: 'POST',
```

```
          credentials: 'include',
          body: formData,
        });
        if (response.ok) {
        } else {
          document.getElementById('AlertaPerfil').innerHTML = '<div
id="AlertaErro" class="alert alert-danger alert-dismissible fade show"
role="alert"><strong>  </strong><button type="button" class="close" data-
dismiss="alert" aria-label="Close"><span aria-
hidden="true">&times;</span></button></div>';
        }
      } catch (error) {
        console.error('Erro de rede:', error);
      }
    }
  });
 }
```

Neste trecho ao ser disparada a função EnviarFotoPerfil, primeiro ele paga o valor de um input do type=”file” e então força no mesmo um evento de click no trecho :

fileInput.click().

EM PRODUÇÃO