**REACTGRAM**

***FRONT-END***

**npm i react-icons –** pegar ícones de ações necessárias;

**react-router-dom –** para rotear as páginas;

**@reduxjs/toolkit –** serve para gerenciar os estados como a contextapi.

**Implementando o Redux**

Instalação do redux de fato:

***npm i react-redux***

**Implementação: *index.js***

- Import Provider

// Redux

import { Provider } from 'react-redux'//similar a contextAPI

- entrega o que está no redux (store manage: guarda dados e compartilha entre a aplicação)

- Onde dados são guardados

//arquivo único que importa os contexts mais variados, só é trabalho um arquivo desse

import { store } from './store'

**arquivo store.js**

***src > store.js***

import { configureStore } from '@reduxjs/toolkit'//importação necessária

export const store = configureStore({//onde contexts estão sendo salvos(deve ser feita a execução da função importada acima 'configureStore')

    reducer:{//onde fica os contexts

    }

})

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <Provider store={store}>{/\*o que será entregue aos componentes, tudo oque está no redux\*/}

      <App />

    </Provider>

  </React.StrictMode>

)

A partir desse envolvimento tudo o que estiver no redux poderá ser trabalhado no app

**- Criação de mais duas pastas**

**services:** lida com o backend, onde requisições com o Fetch serão executadas; É como uma “ponte” a API.

**slices:** ações a serem executadas baseadas nas requisições que tem em serviço. Ex.:

Ação de registro que dispara os seus diversos estados (loading, success..); podem ser feitas manipulações entre esses estados.

**Configs Req. HTTP**

export const api = "http://localhost:5000/api"

export const uploads = "http://localhost:5000/uploads"//para qnd for necessária a inserção de imagens

export const requestConfig = (method, data, token = null, image = null) => {

    //método da reqisição/dados que serão ou foram enviados/token para ver se há autenticação ou não/null-prevendo que user não está autenticado

    let config//muda baseada na requisição

    if (image) {//se existir imagem, objeto será retornado

        config = {

            method, //enviado pelo argumento

            body: data,//dados que vem no argumento data

            headers: {}//precisam existir, mas estarão vazios

        }

    } else if (method === "DELETE" || data === null) {//verificando se método é delete ou não há dados, um exemplo de se utilizar é a função de like, não há dados mas o envio algo tem que ser feito

        config = {

            method,

            headers: {}

        }//só é necessário definir o método e no servidor a função se resolve sozinha

    } else {//quando vem dados

        config = {

            method,

            body: JSON.stringify(data),//dados convertidos para json

            headers: {

                "Context-Type": "application/json"

            }

        }

    }

    if (token) {//se veio token será colocado nos headers//`Bearer ${token}`- para ter string dinâmica

        config.headers.Authorization = `Bearer ${token}` //bearer: trafega um token no Header da requisição e o Basic um usuário e senha.

    }//

    return config

}

//objeto de config. das requisições

**Bearer**

O Bearer trafega um token no Header da requisição e o Basic um usuário e senha.

Lembrando que existem atributos adotados pela comunidade para identificação dos campos (Reserved claims) então o correto é manter esses padrões:  
  
São atributos não obrigatórios (mas recomendados) que são usados na validação do token pelos protocolos de segurança das APIs.

sub (subject) = Entidade à quem o token pertence, normalmente o ID do usuário;  
iss (issuer) = Emissor do token;  
exp (expiration) = Timestamp de quando o token irá expirar;  
iat (issued at) = Timestamp de quando o token foi criado;  
aud (audience) = Destinatário do token, representa a aplicação que irá usá-lo.

**Service de autenticação**

//req. http que lida com a autenticação do usuário

import { api, requestConfig } from "../utils/config"//utilizados para facilitar na hora de fazer requisições

//Register an user - registrar usuário

const register = async (data) => {//funcao async. que recebe dados do usuário

    //formação do request

    const config = requestConfig("POST", data)//config. de requisição, execução de função de config.js; enviando o tipo e os dados da requisição

    try {

        const res = await fetch(api + "/users/register", config).then((res) => res.json()).catch((err) => err)

        //espera uma resposta de um await fetch; url + caminho da requisição; config previamente configurada acima;.then:recebe dados e transforma em um obj js; se algo der errado, cai no catch e erro é retornado

        if (res) {

            localStorage.setItem("user", JSON.stringify(res))

        }//item user do localstorage transformado em res no formato string json

    } catch (error) {

        console.log(error)

    }

}

const authService = {

    register

}

export default authService