**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

**Slice de autenticação**

***slices>authSlice.js***

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'

import authService from '../services/authService'//de onde sai as funções para a execução de coisas do sistema que correspondem ao slice

//sices e services estão diretamente ligados

const user = JSON.parse(localStorage.getItem("user"))//resgatando usuário salvo pelo service na localstorage

const initialState = { //estado inicial

    user: user ? user : null,

    error: false,

    success: false,

    loading: false,

}

//Register an user and sign in - Registrar um usuário e entrar

//exportando as funções

export const register = createAsyncThunk("auth/register",//exporta função e cria ela com createAsyncThunk, 1º argumento:nome, 2º argumento:função //nome segue convenção: auth(nome da entidade que está sendo trabalhada, autenticação) - register(nome atual)

    async (user, ThunkAPI) => {//recebe usuário e ThunkAPI(nos permite utilizar funções extras;ex.: para a execução e identificar um erro da api)

        const data = await authService.register(user)//user vem do componente register//tentativa de fazer cadastro do mesmo

        //check for errors - checagem de erros

        if (data.errors) {//se houver erros

            return ThunkAPI.rejectWithValue(data.errors[0])//rejeitando requisição pois houve algo de errado manualmente

        }//errors vem de backend onde existem diversas mensagens, com o 0 de índice(1º elemento do array), será sempre exibido o 1º erro

        return data//retorno do usuário cadastrado

    }

)

export const authSlice = createSlice({//importando authSlice com funções criadas

    name: "auth",//deve ser renomeado, e assim que ele será chamado na storage - auth

    initialState,//estado inicial

    reducers: {

        reset: (state) => {//todos os estados serão resetados(ficarão como eram no início)

            state.loading = false

            state.error = false

            state.success = false

        },

        extraReducers: (builder) => {//parte das execuções que são feitas na api//builder-construtor: cria ações separadamente

            builder.addCase(register.pending, (state) => {//se a requisição foi enviada mas não obteve nenhuma resposta

                state.loading = true

                state.error = false

                //adicionando casos

            }).addCase(register.fulfilled, (state, action) => {//significa que a operação foi concluída com sucesso.

                state.loading = false

                state.success = true

                state.error = null

                state.user = action.payload//é possível trafegar dados, não apenas mexer nos states; exibição na tela

            }).addCase(register.rejected, (state, action) => {//significa que a operação falhou; tem um action pois tem o dado da mensagem de erro

                state.loading = false//false pois já respondeu algo, logo, não está carregando

                state.error = action.payload//baseado nisso, tem como pegar o erro e exibir na tela

                state.user = null//está tentando cadastrar ou logar, mas não há usuário

            })

        }

    }

})

export const { reset } = authSlice.actions//vem de authSlice.actions(pode haver várias actions)

export default authSlice.reducer

***store.js***

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

import authReducer from "./slices/authSlice"

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

    reducer:{//onde fica os contexts

        auth: authReducer,

    }

})

***pending: estado inicial, nem cumprido nem rejeitado.***

***fulfilled: significa que a operação foi concluída com sucesso.***

***rejected: significa que a operação falhou.***