**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: {

                "Content-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.***

**Conectando formulário ao reducer**

***reactgram/frontend/src/index.js***

import { useSelector, useDispatch } from 'react-redux'

//Redux

import { register, reset } from '../../slices/authSlice'

const dispatch = useDispatch()//nos permite utilizar as funções reducer

const { loading, error } = useSelector((state) => state.auth)//nos permite escolher qual estado e de qual reducer estou usando

dispatch(register(user))

}

//Clean all auth states - limpar todos so states de autenticação(resetar tudo)

useEffect(()=>{

  dispatch(reset())

}, [dispatch])//fazer um useEffect sempre que rola um dispatch faz com que seja possível disparar o reset, assim o reset fica automatizado antes de disparar qualquer função que dispara uma função async

**Criando componente de mensagem**

***reactgram/backend/app.js***

app.use(cors({ credentials: true, origin: "http://localhost:3001" })) //origin - de onde esta vindo a requisição

***reactgram/frontend/src/components/Message.js***

import './Message.css'

const Message = ({msg, type}) => {

  return (

    <div className={`message ${type}`}>

        <p>{msg}</p>

    </div>

  )

}

export default Message

***reactgram/frontend/src/utils/config.js***

method: method,

**Exibindo validação do usuário**

Package.json –

"react-redux": "^8.1.1",

"react-router-dom": "^6.14.1",

"react-scripts": "5.0.1",

**"serialize-javascript": "^6.0.1",**

"web-vitals": "^2.1.4"

},

"scripts": {

***reactgram/frontend/hooks/useAuth.js***

import { useState, useEffect } from 'react'

import { useSelector } from 'react-redux'//para ser possível resgatar dado da storage

export const useAuth = () => {

    const { user } = useSelector((state) => state.auth)//pegando usuário state da autenticação

    const [auth, setAuth] = useState(false)

    const [loading, setLoading] = useState(true)//uma maneira de resgatar dados de login caso usuário recarregue a página

    useEffect(() => {//checando se usuário está logado

        if (user) {//se usuário está setado

            setAuth(true)//está autenticado

        } else {

            setAuth(false)

        }

        setLoading(false)

    }, [user])//useEffect será executado sempre que usuário mudar

    return { auth, loading }

}

**Verificando se usuário está autenticado**

***reactgram/frontend/src/App.js***

import { useAuth } from './hooks/useAuth'

const { auth, loading } = useAuth()

if (loading) {

    return <p>Carregando...</p>

}

<Routes>

    <Route path="/" element={auth ? <Home /> : <Navigate to="/login" />} />

    <Route path="/login" element={!auth ? <Login /> : <Navigate to="/" />} />

    <Route path="/register" element={!auth ? <Register /> : <Navigate to="/" />} />

</Routes>

**Navbar para usuários autenticados**

***reactgram/frontend/src/components/Navbar.js***

import { useState } from 'react'//gerenciar estados

import { useAuth } from '../hooks/useAuth'//para usar autenticação

import { useDispatch, useSelector } from 'react-redux'//para quando tivermos configs de logout por exemplo//para resgatar state do store do reducer

import { useNavigate } from 'react-router-dom'//redirecionar usuários

const Navbar = () => {

    const { auth } = useAuth()

    const { user } = useSelector((state) => state.auth)//pegando usuário state da autenticação

    {

        auth ? (//se usuário tiver autenticado exibe um navlink com o ícone de home

            <>

                <li><NavLink to="/"><BsHouseDoorFill /></NavLink></li>

                {user && (//se usuário estiver logado exibe um navlink com um icone que encaminha para url /users/id-user

                    <li><NavLink to={`/users/${user.\_id}`} /><BsFillCameraFill /></li>//página do usuário

                )}

                <li><NavLink to="/profile" /><BsFillPersonFill /></li>{/\* perfil usuário logado \*/}

                <li><span>Sair</span></li>

            </>

        ) : (

            <>

                <li><NavLink to="/login">Entrar</NavLink></li>

                <li><NavLink to="/register">Cadastrar</NavLink></li>

            </>

        )

    }

}

**Função de logout**

***reactgram/frontend/src/services/authService.js***

//Logout an user - sair de um usuário

const logout = () => {

    localStorage.removeItem("user")//remove user do localstorage

}

const authService = {

    logout

};

***reactgram/frontend/src/slices/authSlice.js***

//Logout an user - sair de um usuário

export const logout = createAsyncThunk("auth/logout", async () => {//auth/logout nome da createAsyncThunk

    await authService.logout()

})//funcao async usa authService(serviço de autenticação)para fazer o logout do usuário atualmente autenticado

    .addCase(logout.fulfilled, (state, action) => {

        state.loading = false

        state.success = true

        state.error = null

        state.user = null

    })

**Implementando logout**

***reactgram/frontend/src/components/Navbar.js***

//Reducer

import { logout, reset } from '../slices/authSlice'

const navigate = useNavigate()

const dispatch = useDispatch()//enviar ou despachar uma ação para um "reducer" em uma arquitetura Flux ou Redux

const handleLogout = () => {

    dispatch(logout())

    dispatch(reset())

    navigate("/login")//como o usuário será desconectado ele é redirecionado para login

}

<li><span onClick={handleLogout}>Sair</span></li>

**Form de login**

***reactgram/frontend/src/pages/Auth/Login.js***

//Components

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

import Message from '../../components/Message'//para exibir mensagens de erro

//Hooks

import { useEffect, useState } from "react"

import { useSelector, useDispatch } from 'react-redux'

//Redux

const Login = () => {

    const [email, setEmail] = useState("")

    const [password, setPassword] = useState("")

    const handleSubmit = (e) => {

        e.preventDefault()

    }

    return (

        <div id="login">

            <h2>ReactGram</h2>

            <p className="subtitle">Faça o login para ver o que há de novo.</p>

            <form onSubmit={handleSubmit}>

                <input type="text" placeholder="E-mail" onChange={(e) => setEmail(e.target.value)} value={email} />

                <input type="password" placeholder="Senha" onChange={(e) => setPassword(e.target.value)} value={password} />

                <input type="submit" placeholder="Entrar" />

            </form>

            <p>Não tem uma conta? <Link to="/register">Clique aqui</Link></p>

        </div>

    )

}

**Service + Slice de login**

***reactgram/frontend/src/services/authService.js***

//Sign in an user - Logar um usuário

const login = async (data) => {//função assíncrona recebe dados

    const config = requestConfig("POST", data)//configs do request config: envio de dados + dados(email e senha)

    try {

        const res = await fetch(api + "/users/login", config)

            .then((res) => res.json())//pega dados e transforma no texto na requisição

            .catch((err) => err)//pega possível erro da requisição e retorna

        if (res) {//se resposta veio

            localStorage.setItem("user", JSON.stringify(res))//pega dados de user na localstorage e converte para string json

        }

        return res

    } catch (error) {

        console.log(error)

    }

}

const authService = {

    login

};

***reactgram/frontend/src/slices/authSlice.js***

//Sign in an user - Logar um usuário

export const login = createAsyncThunk("auth/login", 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.login(user)//user vem do componente login//tentativa de fazer login 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

})

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

        state.loading = true

        state.error = false

        //adicionando casos

    }).addCase(login.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(login.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

    })

**Funcionalidade de login**

***reactgram/frontend/src/pages/Auth/Login.js***

//Redux

import { login, reset } from '../../slices/authSlice'

const { loading, error } = useSelector((state) => state.auth)

console.log(user)

<form>

    ...........

    {!loading && <input type="submit" value="Entrar" />}

    {loading && <input type="submit" disabled value="Aguarde..." />}

    {error && <Message msg={error} type="error" />}

</form>

***reactgram/backend/controllers/UserController.js***

//fazer login do usuário

const login = async (req, res) => {

    const { email, password } = req.body//chaves vindas do body necessárias para fazer login

    const user = await User.findOne({ email })

    // Check if user exists

    if (!user) {

        res.status(404).json({ errors: ["Usuário não encontrado!"] })

        return

    }//se usuário n existir será exibido um err404 + mensagem de erro

    // Check if password matches

    if (!(await bcrypt.compare(password, user.password))) {

        res.status(422).json({ errors: ["Senha inválida!"] })

        return

    }

    // Return user with token

    res.status(200).json({

        \_id: user.\_id,

        profileImage: user.profileImage,

        token: generateToken(user.\_id),

    })

}

const user = await User.findById(mongoose.Types.ObjectId(id)).select("-password")//encontra usuário pelo id

**Estruturando edição do perfil**

***reactgram/frontend/src/App.js***

import EditProfile from "./pages/EditProfile/EditProfile"

<Route path="/profile" element={auth ? <EditProfile /> : <Navigate to="/login" />} />

***reactgram/frontend/src/pages/EditProfile/EditProfile.js***

import './EditProfile.css'

const EditProfile = () => {

    const handleSubmit = (e) => {

        e.preventDefault()

    }

  return (

    <div id="edit-profile">

        <h2>Edite seus dados</h2>

        <p className="subtitle">Adicione uma imagem de perfil e conte mais sobre você...</p>

        {/\* preview da imagem \*/}

        <form onSubmit={handleSubmit}>

        <input type="text" placeholder="Nome" />

        <input type="email" placeholder="E-mail" disabled />

        <label>

            <span>imagem do Perfil: </span>

            <input type="file" />

        </label>

        <label>

            <span>Bio: </span>

            <input type="text" placeholder="Descrição do perfil" />

        </label>

        <label>

            <span>Quer alterar sua senha?: </span>

            <input type="password" placeholder="Digite sua nova senha:" />

        </label>

        <input type="submit" value="Atualizar" />

        </form>

    </div>

  )

}

export default EditProfile

**- Usuário**

**Reducer – usuário**

*reactgram/frontend/src/slices/userSlice.js*

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

const initialState = {//estado inicial com user vazio, mensage é nula

    user: {},

    error: false,

    success: false,

    loading: false,

    message: null

}

//funcoes

export const userSlice = createSlice({//reducer (ficará no store)

    name: "user",

    initialState,

    reducers: {

        resetMessage: (state) => {

            state.message = null

        }//reseta msg

    }

})

export const { resetMessage } = userSlice.actions//função de resetar mensagem

export default userSlice.reducer//reducer

*reactgram/frontend/src/store.js*

import userReducer from "./slices/userSlice"//usuario

export const store = configureStore({

    reducer: {

        user: userReducer,

    }

})

**Resgatando dados de usuário autenticado**

*reactgram/frontend/src/services/userService.js*

import { api, requestConfig } from '../utils/config'

//get user details - obter detalhes do usuário

const profile = async (data, token) => {

    const config = requestConfig("GET", data, token)//config da requisição passando a função requestConfig recebendo método get e os dados e o token

    try {

        const res = await fetch(api + "/users/profile", config)//perfil do usuário logado(baseado no token)

            .then((res) => res.json)//será transformada em obj javascript

            .catch((err) => err)//se houver algum erro

        return res

    } catch (error) {

        console.log(error)

    }

}

const userService = {

    profile

}

export default userService

**Slice – perfil do usuário**

*reactgram/frontend/src/slices/userSlice.js*

import userService from '../services/userService'

//Get user details

export const profile = createAsyncThunk(

    "user/profile",

    async (user, thunkAPI) => {

        const token = thunkAPI.getState().auth.user.token

        const data = await userService.profile(user, token)

        return data

    }

)

export const userSlice = createSlice({

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

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

            state.loading = true

            state.error = null

        })

            .addCase(profile.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

            })

    },

})

**Preenchendo form de edição com dados do usuário**

*reactgram/frontend/src/pages/EditProfile/EditProfile.css*

#edit-profile{}

*reactgram/frontend/src/pages/EditProfile/EditProfile.js*

import { uploads } from "../../utils/config"//diretório onde são feitos uploads de imagem

//hooks

import { useEffect, useState } from 'react'//react

import { useSelector, useDispatch } from 'react-redux'//redux

//redux - slices

import { profile, resetMessage } from '../../slices/userSlice'

//components

import Message from '../../components/Message'

const EditProfile = () => {

    const dispatch = useDispatch()

    const { user, message, error, loading } = useSelector((state) => state.user)//estados vindos do user

    const [name, setName] = useState("")

    const [email, setEmail] = useState("")

    const [password, setPassword] = useState("")

    const [profileImage, setProfileImage] = useState("")

    const [bio, setBio] = useState("")

    const [previewImage, setPreviewImage] = useState("")

    //load user data - carregar dados do usuário

    useEffect(() => {

        dispatch(profile())

    }, [dispatch])

    //fill form with user data - preencher formulário com dados do usuário

    useEffect(() => {

        if (user) {

            setName(user.name)

            setEmail(user.email)

            setBio(user.bio)

        }

    }, [user])

    const handleSubmit = async (e) => {

        e.preventDefault()

    }

    <form onSubmit={handleSubmit}>

        <input type="text" placeholder="Nome" onChange={(e) => setName(e.target.value)} value={name} />

        <input type="email" placeholder="E-mail" disabled value={email} />

        <label>

            <span>imagem do Perfil: </span><input type="file" /></label>

        <label>

            <span>Bio: </span><input type="text" placeholder="Descrição do perfil" onChange={(e) => setBio(e.target.value)} value={bio} />

        </label>

        <label>

            <span>Quer alterar sua senha?: </span><input type="password" placeholder="Digite sua nova senha:" onChange={(e) => setPassword(e.target.value)} value={password} />

        </label>

        <input type="submit" value="Atualizar" />

    </form>

*reactgram/frontend/src/slices/userSlice.js –* ***Anotações***

//Get user details - Obter detalhes do usuário

export const profile = createAsyncThunk(

    "user/profile",//renomeando funcao

    async (user, thunkAPI) => {//thunkAPI vai auxiliar para pegar o state

        const token = thunkAPI.getState().auth.user.token//podem ser obtidos dados do redux e do thunkAPI/.user.: salvo no authSlice

        const data = await userService.profile(user, token)//obtemos método profile do userService com user e token

        console.log(data);

        return data

    }

)

**Exibindo imagem de preview**

*reactgram/frontend/src/pages/EditProfile/EditProfile.js*

const handleFile = (e) => {//setar valores de imagem

    //image preview

    const image = e.target.files[0];

    setPreviewImage(image)

    //update image state - atualizar estado da imagem

    setProfileImage(image)

}

return (

    <>

        {(user.profileImage || previewImage) && (

            <img className='profile-image'

                src={previewImage ? URL.createObjectURL(previewImage) : `${uploads}/users/${user.profileImage}`} alt={user.name}//url.create.....:transformando imagem em html

            />//exibe dados da previewImage ou a imagem(caminho da imagem)

        )}{/\*usuario tem imagem ou houve mudanças em previewImage\*/}

        <label>

            <span>imagem do Perfil: </span><input type="file" onChange={handleFile} /></label>

        <label> </label>

    </>

)

**User Update**

*reactgram/frontend/src/services/userService.js*

//Update user details - Atualizar detalhes do usuário

const updateProfile = async (data, token) => {//token-permite a autenticação/data-preciso de dados para atualização do usuário

    const config = requestConfig("PUT", data, token, true)//config requisição//put: atualização completa de um registro//true: req. pode conter imagens

    try {

        const res = await fetch(api + "/users/", config)//espera resposta de um await fecth api + url(/users/-barra na frente pois está trabalhando com atualização)+config(configs necessárias para atualizar com sucesso)

            .then((res) => res.json())

            .catch((err) => err)

    } catch (error) {

        console.log(error)

    }

}

const userService = {

    updateProfile

}

**Update Slice**

*reactgram/frontend/src/slices/userSlice.js*

//update user details - atualizar detalhes do usuário

export const updateProfile = createAsyncThunk("user/update", async (user, thunkAPI) => {

    const token = thunkAPI.getState.auth.user.token

    const data = await userService.updateProfile(user, token)

    //check for erros - verificar se há erros

    if (data.errors) {

        return thunkAPI.rejectWithValue(data.errors[0])

    }

    return data

})

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

        state.loading = true

        state.error = false

        //adicionando casos

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

        state.loading = false;

        state.success = true;

        state.error = null;

        state.user = action.payload;

        state.message = "Usuário atualizado com sucesso!";//é possível trafegar dados, não apenas mexer nos states exibição na tela

    }).addCase(updateProfile.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

    });

**Finalizando update**

*reactgram/backend/controllers/UserController.js -* ***Ajuste***

const user = await User.findById(id).select("-password")//encontra usuário pelo id

*reactgram/frontend/src/pages/EditProfile/EditProfile.js*

useEffect(() => {

    if (user) {//se tiver usuário, retornará os dados:

        setName(user.name);

        setEmail(user.email);

        setBio(user.bio);

    }

}, [user]);//sempre que usuário mudar useEffect é disparado

const handleSubmit = async (e) => {

    e.preventDefault();//utilizada para prevenir o comportamento padrão de um evento.

    //gather user data form states - reunir os estados do formulário de dados do usuário

    const userData = {

        name,

    };

    //só envia o necessário

    if (profileImage) {

        userData.profileImage = profileImage;

    }

    if (bio) {

        userData.bio = bio;

    }

    if (password) {

        userData.password = password;

    }

    //build form data - criar dados do formulário

    const formData = new FormData()//instanciando objeto

    const userFormData = Object.keys(userData).forEach((key) => formData.append(key, userData[key]));

    //loop em todas as chaves que serão enviadas//foreach- executa a função em cada elemento de um array//.append-coloca novo valor no final dos valores existente.

    formData.append("user", userFormData);

    await dispatch(updateProfile(formData));//dipstach na updateProfile passando o valor de userFormData

    setTimeout(() => {

        dispatch(resetMessage());

    }, 2000);

};

{ !loading && <input type="submit" value="Atualizar" /> }

{ loading && <input type="submit" disabled value="Aguarde..." /> }

{ error && <Message msg={error} type="error" /> }

{ message && <Message msg={message} type="success" /> }

</form >

**Criando página de postagem de fotos**

*reactgram/frontend/src/App.js*

<Route path="/users/:id" element={auth ? <Profile /> : <Navigate to="/login" />} />

*reactgram/frontend/src/pages/Profile/Profile.js*

import './Profile.css'

import { uploads } from '../../utils/config'

//components

import Message from '../../components/Message'

import { Link } from "react-router-dom"

import { BsFillEyeFill, BsPencilFill, BsPencil, BsXLg } from 'react-icons/bs'

//hooks

import { useState, useEffect, useRef } from "react"

import { useSelector, useDispatch } from "react-redux"

import { useParams } from "react-router-dom"

//redux

const Profile = () => {

    return (

        <div>Profile</div>

    )

}

export default Profile

**Página de postagem de fotos**

*reactgram/frontend/src/App.js-* **Ajuste**

<Route path="login" element={!auth ? <Login /> : <Navigate to="/" />} />

<Route path="register" element={!auth ? <Register /> : <Navigate to="/" />} />

*reactgram/frontend/src/services/userService.js*

//get user details - obter detalhes do usuário

const getUserDetails = async (id) => {//

    const config = requestConfig("GET")

    try {

        const res = await fetch(api + "/users/: id, config")

            .then((res) => res.json())

            .catch((err) => err)

    } catch (error) {

        console.log(error)

    }

}

const userService = {

    getUserDetails

}

**Slice + Service – Resgatando dados do usuário pelo id**

*reactgram/frontend/src/slices/userSlice.js*

//get user details - obter detalhes do usuário

export const getUserDetails = createAsyncThunk("user/get", async (id, thunkAPI) => {

    //não precisa ser feita a validação com o token pois já foi feita no back-end

    const data = await userService.getUserDetails(id)

    return data

})

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

        state.loading = true

        state.error = null

    })

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

        state.loading = false

        state.success = true

        state.error = null

        state.user = action.payload

    })

**Exibindo dados no perfil**

*reactgram/frontend/src/pages/Profile/Profile.js*

const Profile = () => {

    const { id } = useParams()//id da url de quando um usuário entra no perfil de outro

    const dispatch = useDispatch()//do hook do redux, para usar por exemplo, função de chamar dados

    const { user, loading } = useSelector((state) => state.user)//usuário que outra pessoa entrou no perfil

    const { user: userAuth } = useSelector((state) => state.auth)//usuário autenticado

    //photo

    //load user data - carregar dados do usuário

    useEffect(() => {

        dispatch(getUserDetails(id))//detalhes e id do usuário

    }, [dispatch, id])

    if (loading) {

        return <p>Carregando...</p>

    }

*reactgram/frontend/src/pages/Profile/Profile.css*

#profile {

    width: 50%;

    margin: 0 auto;

  }

  #profile-header {

    display: flex;

    align-items: center;

    flex-wrap: wrap;

    padding: 1em;

    border-bottom: 1px solid #363636;

  }

    #profile-header img {

    width: 100px;

    height: 100px;

    border-radius: 50%;

    margin-right: 2em;

  }

**Form de post de foto**

*reactgram/frontend/src/pages/Profile/Profile.js*

// new form and edit form refs - novo formulário e editar refs de formulário

const newPhotoForm = useRef()

const editPhotoForm = useRef()

const submitHandle = (e) => {

    e.preventDefault()

}

if (loading) {

    return <p>Carregando...</p>

}

const Profile = () => {

    <p>{user.bio}</p>

        </div >

      </div >

    { id === userAuth.\_id && (

        <>

            <div className="new-photo" ref={newPhotoForm}>

                <h3>Compartilhe algum momento seu:</h3>

                <form onSubmit={submitHandle}>

                    <label>

                        <span>Título para a foto</span>

                        <input type="text" placeholder="Insira um título" />

                    </label><label>

                        <span>Imagem: </span>

                        <input type="file" />

                    </label>

                    <input type="submit" value="Postar" />

                </form>

            </div>

        </>

    )}

    </div >

**Slice + Service - inserir uma foto**

*reactgram/frontend/src/services/photoService.js*

import { api, requestConfig } from '../utils/config'

//publish an user photo - publicar uma foto de usuário

const publishPhoto = async (data, token) => {//token - função é restrita para quem está logado, por isso a necessidade

    const config = requestConfig("POST", data, token, true)//true(há imagem envolvida)

    try {

        const res = await fetch(api + '/photos', config)//url a ser atingida com respectivos valores

            .then((res) => res.json())

            .catch((err) => err)

        return res//retorna resposta

    } catch (error) {

        console.log(error)

    }

}

const photoService = {

    publishPhoto,

}

export default photoService//para futuramente acessar funções do objeto

*reactgram/frontend/src/slices/photoSlice.js*

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

import photoService from '../services/photoService'

const initialState = {//estado inicial de fotos

    photos: [],

    photo: {},//foto individual

    error: false,

    success: false,

    loading: false,

    message: null

}

//publish user photo - publicar foto do usuário

export const publishPhoto = createAsyncThunk("photo/publish", async (photo, thunkAPI) => {

    const token = thunkAPI.getState().auth.user.token//obter token

    const data = await photoService.publishPhoto(photo, token)//foto e autenticação do usuário

    //check for erros - verificar se há erros

    if (data.errors) {

        return thunkAPI.rejectWithValue(data.errors[0])

    }//rejeita e exibe no formulário

    return data

})

export const photoSlice = createSlice({

    name: "photo",

    initialState,

    reducers: {

        resetMessage: (state) => {

            state.message = null

        },

    },

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

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

            state.loading = true

            state.error = null

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

            state.loading = false

            state.success = true

            state.error = null

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

            state.photos.unshift(state.photo)

            state.message = "Foto publicada com sucesso!"

        }).addCase(publishPhoto.rejected, (state, action) => {

            state.loading = false

            state.error = action.payload

            state.photo = {}

        })

    }

})

export const { resetMessage } = photoSlice.actions

export default photoSlice.reducer

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

import photoService from '../services/photoService'

const initialState = {//estado inicial de fotos

    photos: [],

    photo: {},//foto individual

    error: false,

    success: false,

    loading: false,

    message: null

}

//publish user photo - publicar foto do usuário

export const publishPhoto = createAsyncThunk("photo/publish", async (photo, thunkAPI) => {

    const token = thunkAPI.getState().auth.user.token//obter token

    const data = await photoService.publishPhoto(photo, token)//foto e autenticação do usuário

    //check for erros - verificar se há erros

    if (data.errors) {

        return thunkAPI.rejectWithValue(data.errors[0])

    }//rejeita e exibe no formulário

    return data

})

export const photoSlice = createSlice({

    name: "photo",

    initialState,

    reducers: {

        resetMessage: (state) => {

            state.message = null

        },

    },

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

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

            state.loading = true

            state.error = null

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

            state.loading = false

            state.success = true

            state.error = null

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

            state.photos.unshift(state.photo)

            state.message = "Foto publicada com sucesso!"

        }).addCase(publishPhoto.rejected, (state, action) => {

            state.loading = false

            state.error = action.payload

            state.photo = {}

        })

    }

})

export const { resetMessage } = photoSlice.actions

export default photoSlice.reducer

*reactgram/frontend/src/store.js*

import photoReducer from "./slices/photoSlice"//photo

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

    reducer: {//onde fica os contexts

        photo: photoReducer

    }

})

**Inserir foto de usuário**

*reactgram/frontend/src/pages/Profile/Profile.js*

import { publishPhoto, resetMessage } from '../../slices/photoSlice'

const { photos, loading: loadingPhoto, message: messagePhoto, error: errorPhoto } = useSelector((state) => state.photo)//seleciona estados da photo

//renomeando porque funções já existem

const [title, setTitle] = useState("")

const [image, setImage] = useState("")

const handleFile = (e) => {//setar valores de imagem

    //image preview

    const image = e.target.files[0];

    setImage(image);

}

const submitHandle = (e) => {

    const photoData = {

        title,

        image

    }//dados da foto

    //build form data - construir dados de formulário

    const formData = new FormData()

    const photoFormData = Object.keys(photoData).forEach((key) => formData.append(key, photoData[key]))//Object.keys()-loop em todas as chaves do objeto

    //.forEach - executa uma função fornecida uma vez para cada elemento do array/.append - coloca novo valor no final dos valores existente.

    formData.append("photo", photoFormData)

    dispatch(publishPhoto(formData))

    setTitle("")

    setTimeout(() => {

        dispatch(resetMessage());

    }, 2000);

}

<form onSubmit={submitHandle}>

    <label>

        <span>Título para a foto</span>

        <input type="text" placeholder="Insira um título" onChange={(e) => setTitle(e.target.value)} value={title || ""} />

    </label>

    <label>

        <span>Imagem: </span>

        <input type="file" onChange={handleFile} />

    </label>

    {!loadingPhoto && <input type="submit" value="Postar" />}

    {loadingPhoto && <input type="submit" disabled value="Aguarde..." />}

</form>

          </div >

    { errorPhoto && <Message msg={errorPhoto} type={"error"} />}

{ messagePhoto && <Message msg={messagePhoto} type={"success"} /> }

**Service + Slice de foto do usuário**

*reactgram/frontend/src/services/photoService.js*

//get user photos - obter fotos do usuário

const getUserPhotos = async (id, token) => {//id do usuário, pois a foto vai ser pega de um user específico

    const config = requestConfig("GET", null, token)//uma url precisa ser atingida com isso

    try {

        const res = await fetch(api + "/photos/user/" + id, config)

            .then((res) => res.json())

            .catch((err) => err)

    } catch (error) {

        console.log(error)

    }

}

const photoService = {//exportando funções

    getUserPhotos,

}

//get user photos - obter fotos do usuário

export const getUserPhotos = createAsyncThunk("photo/userphotos", async (id, thunkAPI) => {//nome + ID e thunkAPI

    const token = thunkAPI.getState().auth.user.token//user validation for being a private method

    const data = await photoService.getUserPhotos(id, token)

    return data//retorna dados

})

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

        state.loading = true

        state.error = null

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

        state.loading = false

        state.success = true

        state.error = null

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

    })

**Exibindo foto no perfil do usuário**

*reactgram/frontend/src/pages/Profile/Profile.js*

import { publishPhoto, resetMessage, getUserPhotos } from '../../slices/photoSlice'

useEffect(() => {

    dispatch(getUserPhotos(id))

}, [dispatch, id])

    <>

    < div className="user-photos" >

        <h2>Fotos publicadas:</h2>

        <div className="photos-container">

            {photos && photos.map((photo) => (//se fotos existirem, são mapeadas e cada uma é nomeada como "photo"

                <div className="photo" key={photo.\_id}>

                    {photo.image && (<img src={`${uploads}/photos/${photo.image}`} alt={photo.titl} />)}{/\* verificando se photo chegou, se sim, exibe imagem\*/}

                    {id === userAuth.\_id ? (

                        <p>actions</p>

                    ) : (<Link className="btn" to={`/photos/${photo.\_id}`}>Ver</Link>)}

                </div>

            ))}

            {photos.length === 0 && <p>Ainda não há fotos publicadas</p>}

        </div>

    </ >

**CSS perfil do usuário**

*reactgram/frontend/src/pages/Profile/Profile.css*

.new-photo {

  padding: 1em;

  border-bottom: 1px solid #363636;

}

.photos-container {

  display: flex;

  flex-wrap: wrap;

}

.photo {

  width: 32%;

  margin: 0.3%;

}

.photo img {

  width: 100%;

}

.actions {

  display: flex;

  justify-content: space-around;

  padding: 10px;

}

.actions svg {

  cursor: pointer;

}

.edit-photo {

  margin-bottom: 1em;

}

.edit-photo img {

  width: 100%;

  margin-bottom: 1em;

}

**Service + Slice de fotos excluídas**

*reactgram/frontend/src/services/photoService.js*

//delete a photo - excluir uma foto

const deletePhoto = async (id, token) => {//id e token: identificação do usuário

    const config = requestConfig("DELETE", null, token)

    try {

        const res = await fetch(api + '/photos/' + id, config)

            .then((res) => res.json())

            .then((err) => err)

        return res

    } catch (error) {

        console.log(error)

    }

}

const photoService = {//exportando funções

    deletePhoto,

}

*reactgram/frontend/src/slices/photoSlice.js*

//delete a photo - remover uma foto

export const deletePhoto = createAsyncThunk("photo/delete", async (id, thunkAPI) => {

    const token = thunkAPI.getState().auth.user.token//user validation for being a private method

    const data = await photoService.deletePhotos(id, token)

    //check for erros - verificar se há erros

    if (data.errors) {

        return thunkAPI.rejectWithValue(data.errors[0])

    }//rejeita e exibe no formulário

    return data

})

export const photoSlice = createSlice({

    name: "publish",

},

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

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

        state.loading = true;

        state.error = null;

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

        state.loading = false;

        state.success = true;

        state.error = null;

        state.photos = state.photos.filter((photo) => {

            return photo.\_id !== action.payload.id

        })

        state.message = action.payload.message

    }).addCase(deletePhoto.rejected, (state, action) => {

        state.loading = false;

        state.error = action.payload;

        state.photo = {};

    })

 }

**Finalizando função de exclusão de foto**

*reactgram/frontend/src/pages/Profile/Profile.js*

import { publishPhoto, resetMessage, getUserPhotos, deletePhoto } from '../../slices/photoSlice'

setTimeout(() => {

    dispatch(resetMessage());

  }, 2000);

resetComponentMessage()

//delete a photo - exclusão de uma foto

const handleDelete = (id) => {

dispatch(deletePhoto(id))

resetComponentMessage()

}

<BsXLg onClick={() => handleDelete(photo.\_id)} />{/\* se n tivesse sido usada a arrow function, seria executado assim que aparecesse na tela\*/}

</div>

**Service + Slice – Update photo**

*reactgram/frontend/src/pages/Profile/Profile.js -* **Ajuste**

 // Reset component message

 function resetComponentMessage() {

    setTimeout(() => {

      dispatch(resetMessage());

    }, 2000);

  }

*reactgram/frontend/src/services/photoService.js*

try {

    const res = await fetch(api + "/photos/" + id, config)

        .then((res) => res.json())

        .catch((err) => err);

    return res;

} catch (error) {

    console.log(error);

}

};

//update a photo - atualizar uma foto

const updatePhoto = async (data, id, token) => {//dados, id e token por ser uma função privada

    const config = ("PUT", data, token)//put-atualização de dados, data e token para configurar o request

    updatePhoto,

}

*reactgram/frontend/src/slices/photoSlice.js*

const data = await photoService.deletePhoto(id, token)

//check for erros - verificar se há erros

if (data.errors) {

    return thunkAPI.rejectWithValue(data.errors[0])

}//rejeita e exibe no formulário

return data

})

//update a photo - atualizar

export const updatePhoto = createAsyncThunk("photo/update", async (photoData, thunkAPI) => {

    const token = thunkAPI.getState().auth.user.token//user validation for being a private method

    const data = await photoService.updatePhoto({ title: photoData.title }, photoData.id, token)

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

            state.loading = true;

            state.error = null;

        })

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

            state.loading = false;

            state.success = true;

            state.error = null;

            state.photos.map((photo) => {

                if (photo.\_id !== action.payload.photo.\_id) {

                    return photo.title = action.payload.photo.title

                }

                return photo

            })

            state.message = action.payload.message

        })

        .addCase(updatePhoto.rejected, (state, action) => {

            state.loading = false;

            state.error = action.payload;

            state.photo = {};

        })

**Finalizando edição de foto**

*reactgram/frontend/src/pages/Profile/Profile.js*

import { publishPhoto, resetMessage, getUserPhotos, deletePhoto, updatePhoto } from '../../slices/photoSlice'

//states de inclusão

const [title, setTitle] = useState("")

const [image, setImage] = useState("")

//states de edição

const [editId, setEditId] = useState("")

const [editImage, setEditImage] = useState("")

const [editTitle, setEditTitle] = useState("")

//show or hide forms - mostrar ou ocultar formulários

const hideOrShowForms = () => {

    newPhotoForm.current.classList.toggle("hide")//se estiver sendo exibido hide será adicionado, caso o contrário hide sai e esconde form

    editPhotoForm.current.classList.toggle("hide")

}

//update a photo - atualizar uma foto

const handleUpdate = (e) => {

    e.preventDefault()

    const photoData = {

        title: editTitle,

        id: editId

    }//dados da foto para fazer atualização

    dispatch(updatePhoto(photoData))

    resetComponentMessage()

}

//open edit form - abrir edição de formulário

const handleEdit = (photo) => {

    if (editPhotoForm.current.classList.contains("hide")) {//verificando se editPhotoForm está com a class hide ou não

        hideOrShowForms()

    }

    setEditId(photo.\_id)

    setEditTitle(photo.title)

    setEditImage(photo.image)

    //preenchendo form com as informações da imagem

}

const handleCancelEdit = (e) => {

    hideOrShowForms()

}

<div className="edit-photo hide" ref={editPhotoForm}>{/\*hide pois começa escondida, só será exibida quando usuário clicar que quer editar uma foto\*/}

    <p>Editando:</p>

    {editImage && (//checando se editImage está presente

        <img src={`${uploads}/photos/${editImage}`} alt={editTitle} />

    )}

    <form onSubmit={handleUpdate}>

        <input type="text" placeholder="Insira o novo título" onChange={(e) => setEditTitle(e.target.value)} value={editTitle} />

        <input type="submit" value="Atualizar" />

        <button className="cancel-btn" onClick={handleCancelEdit}>Cancelar edição</button>

    </form>

    <BsPencilFill onClick={() => handleEdit(photo)} />

*reactgram/frontend/src/services/photoService.js*

const config = ***requestConfig***("PUT", data, token)

*reactgram/frontend/src/slices/photoSlice.js*

const data = await photoService.updatePhoto({ title: photoData.title }, photoData.id, token)//atualizando dados de acordo com os dados da foto

if (photo.\_id === action.payload.photo.\_id) {//verificando id da foto

    return (photo.title = action.payload.photo.title)

}//atualizando foto com oq há no frontend para não ser necessário fazer uma nova requisição

**Criando página de foto individual**

*reactgram/frontend/src/App.js*

import Photo from "./pages/Photo/Photo";

<Route path="/photos/:id" element={auth ? <Photo /> : <Navigate to="/login" />} />

*reactgram/frontend/src/pages/Photo/Photo.js*

import './Photo.css'

import { uploads } from '../../utils/config'//onde contém as imagens

//components

import Message from '../../components/Message'

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

//hooks

import { useEffect, useState } from 'react'

import { useSelector, useDispatch } from 'react-redux'

import { useParams } from 'react-router-dom'

//redux

const Photo = () => {

    return (

        <div>Photo</div>

    )

}

export default Photo

**Service + Slice de foto por id**

*reactgram/frontend/src/services/photoService.js*

//get a photo by id - obter foto por id

const getPhoto = async (id) => {

    const config = requestConfig("GET",)//configuração da requisição

    try {

        const res = await fetch(api + "/photos/" + id, config)//id vem pela url

            .then((res) => res.json())

            .catch((err) => err)

        return res

    } catch (error) {

        console.log(error)

    }

}

const photoService = {//exportando funções

    getPhoto

}

*reactgram/frontend/src/slices/photoSlice.js*

//get photo by id - obter foto por id

export const getPhoto = createAsyncThunk("photo/getphoto", async (id) => {

    const data = await photoService.getPhoto(id)

    return data

})

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

        state.loading = true;

        state.error = false;

    })

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

        state.loading = false;

        state.success = true;

        state.error = null;

        state.photos = action.payload;

    })