

Desenvolvimento de Software para WEB

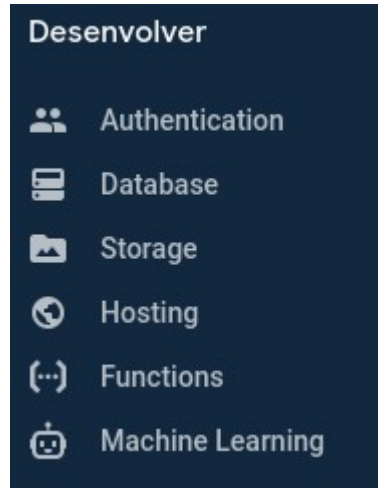
Redux-Firebase-Authentication

Introdução

- Neste projeto, iremos implementar uma aplicação de **cadastro (signup)**, **login (signin)** e **logout**, fazendo uso do firebase authentication para gerenciar usuários e senhas.
- O seu projeto irá necessitar das seguintes bibliotecas:
 - **"firebase": "^7.14.4"** → *Acesso à API do Firebase. É necessário ter conta no Google*
 - **"redux": "^4.0.5"** → *Redux “puro”*
 - **"react-redux": "^7.2.0"** → *“Cola” entre o React e o Redux*
 - **"react-redux-firebase": "^3.5.1"** → *“Cola” entre o React, Redux e Firebase*
 - **"react-router-dom": "^5.1.2"** → *Gerenciador de rotas*
 - **"redux-thunk": "^2.3.0"** → *Middleware responsável por ações assíncronas*







Firebase Authentication

- Crie um projeto no Firebase e vá em **Authentication**:



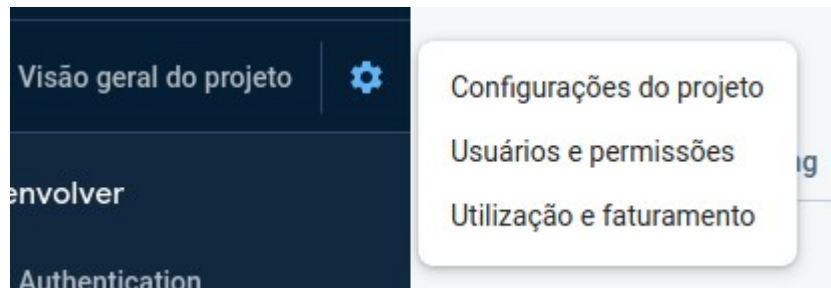
Firebase Authentication

- Na forma de login, escolha por **e-mail/senha**

Provedores de login	
Provedor	Status
 E-mail/senha	Ativado
 Smartphone	Desativado
 Google	Desativado
 Play Games	Desativado
 Game Center Beta	Desativado
 Facebook	Desativado

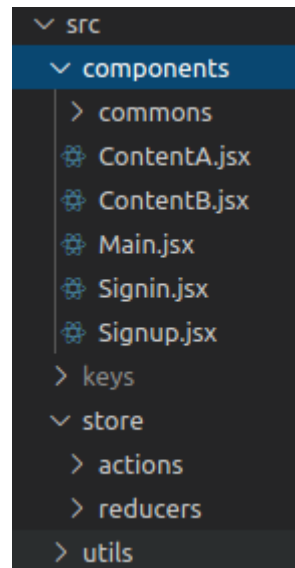
Firebase Authentication

- Em **configurações do projeto**, crie uma aplicação WEB para acessar o projeto e copie a chave de autenticação a ser colocada no arquivo javascript do seu projeto.



O Projeto React

- Uma vez instaladas as bibliotecas, crie o seguinte esquema de pastas:



Conexão com o Firebase

- Na pasta **keys**, crie o arquivo **firebase_key.js**:

```
const firebase_key = {  
  apiKey: "SDdsddkse8dsd-WlGeik_sdssdsdeeEEssdsdsd",  
  authDomain: "pet-redux-auth.firebaseio.com",  
  databaseURL: "https://pet-dsdsdsds-dsd.dsdddd.com",  
  projectId: "pet-sdsdsd-authksdsdeeedusyd",  
  StorageBucket: "????",  
  messagingSenderId: "0000000000",  
  appId: "1sldjslue!!!sdssdsdsd:webskdhksds23033fa13315219d685e7a"  
}  
  
export default firebase_key
```

Conexão com Firebase

- Na pasta **utils**, crie o arquivo **firebase.js**

```
import firebase from 'firebase/app'
```

```
import 'firebase/auth'
```

```
import firebase_key from '../keys/firebase_key'
```

```
firebase.initializeApp(firebase_key)
```

```
export default firebase
```


Conexão com Firebase

- Note que o arquivo `firebase.js` usa o arquivo `firebase_key.js`, para poder fazer a conexão com o Firebase.
- Você deve ter criado um projeto no Firebase e um aplicação WEB, gerando assim a chave de conexão.

Reducers

- Os **reducers**, armazenam o estado geral da aplicação de uma determinada entidade do domínio da aplicação. No nosso caso, iremos armazenar a **mensagem gerada** pelas operações de cadastro, login e logout.
- Cada reducer é um função que recebe como parâmetro um **state** (estado geral anterior) e uma **action** (ação).
- Na pasta reducers iremos criar:
 - **authReducer.js** (reduzidor responsável pela autenticação)
 - **Index.js** (estado geral da aplicação, a combinação de todos os redutores)

authReducer.js

```
import { SIGNUP_SUCCESS, SIGNUP_ERROR, SIGNIN_SUCCESS,  
SIGNIN_ERROR, SIGNOUT_SUCCESS, SIGNOUT_ERROR }  
from '../actions/actionTypes'
```

```
const INITIAL_STATE = {  
  authMsg: null,  
  user: ""  
}
```

Para este exemplo, iremos armazenar apenas a mensagem de resultado de uma ação "authMsg" e "user", o qual armazenará o e-mail logado.

```
export default function (state = INITIAL_STATE, action) {  
  console.log('action: ' + action.type + ' ' + state.user)  
  switch (action.type) {  
    case SIGNUP_SUCCESS:  
      return {  
        ...state,  
        authMsg: action.payload.authMessage,  
        user: action.payload.userMail,  
      }  
    case SIGNUP_ERROR:  
      return {  
        ...state,  
        authMsg: action.payload.authMessage  
      }  
    case SIGNIN_SUCCESS:  
      return {  
        ...state,  
        authMsg: action.payload.authMessage,  
        user: action.payload.userMail,  
      }  
  }  
}
```

Atenção! O arquivo actionTypes.js deve ser Criado antes.

```
    case SIGNIN_ERROR:  
      return {  
        ...state,  
        authMsg: action.payload.authMessage  
      }  
    case SIGNOUT_SUCCESS:  
      return {  
        user: null,  
        authMsg: action.payload.authMessage  
      }  
    case SIGNOUT_ERROR:  
      return {  
        ...state,  
        authMsg: action.payload.authMessage  
      }  
    default:  
      return state  
  }  
}
```

Atenção! Cada "case" trata de um tipo de action diferente. Lembre-se: toda vez que o usuário der um refresh na página (F5), o estado é resetado para o inicial.

index.js

```
import {combineReducers} from 'redux'  
  
import authReducer from './authReducer'  
import { firebaseReducer } from 'react-redux-firebase'  
  
/* ESTADO GERAL DA APLICAÇÃO */  
export default combineReducers({  
  firebaseReducer,  
  authReducer  
})
```

*Temos dois redutores (reducers) declarados neste arquivo: “**firebaseReducer**”, definido pela biblioteca react-redux-firebase; e “**authReducer**”, criado por nós para guardar informações do usuário logado.*

Actions

- As ações são disparadas de acordo com **eventos** da interface do usuário (navegador).
- As ações podem **alterar o estado geral da aplicação** (definido em reducers/index.js), acionando os redutores (que escolhem qual ação irão computar).
- Cada ação tem um tipo (**type**) e um **payload** (informações do novo estado).
- No nosso projeto, dentro da pastas **actions**, temos:
 - actionTypes.js (arquivo simples de constantes)
 - authActionCreator.js (criador de ações)
- Na verdade, as ações são **objetos simples em javascript** ({type,payload}). O arquivo **authActionCreator** é formado por três funções (action creators) responsáveis em retornar uma ação. São elas:
 - signin (email,password,callback) → A função de login
 - signup (email,password,callback) → A função de cadastro
 - signout () → A função de logout

actionType.js

```
export const SIGNUP_SUCCESS = 'SIGNUP_SUCCESS'  
export const SIGNUP_ERROR = 'SIGNUP_ERROR'
```

```
export const SIGNIN_SUCCESS = 'SIGNIN_SUCCESS'  
export const SIGNIN_ERROR = 'SIGNIN_ERROR'
```

```
export const SIGNOUT_SUCCESS = 'SIGNOUT_SUCCESS'  
export const SIGNOUT_ERROR = 'SIGNOUT_ERROR'
```

Define um conjunto de constantes que facilita e padroniza o uso das ações, evitando erros de grafia.

```
import {SIGNUP_SUCCESS,SIGNUP_ERROR,SIGNIN_SUCCESS,SIGNIN_ERROR,SIGNOUT_SUCCESS,SIGNOUT_ERROR} from
'./actionTypes'
import firebase from '../utils/firebase'
```

```
export const signup = (email,password,callback) => {
  return dispatch =>{
    try{
      firebase
      .auth()
      .createUserWithEmailAndPassword(email,password)
      .then(
        ()=>{
          firebase.auth().onAuthStateChanged(
            (user)=>{
              if(user){
                dispatch({
                  type:SIGNUP_SUCCESS,
                  payload: {
                    authMessage: `Cadastro efetuado com sucesso!`,
                    userMail: user.email}
                })
                callback()
              }else{
                dispatch({
                  type:SIGNUP_ERROR,
                  payload: {authMessage: `Não foi possível conectar`}
                })
                callback()
              }
            }
          )//firebase.auth().onAuthStateChanged(
        )//se deu certo
      )
    }
  }
}
```

authActionCreator.js

signup

Atenção! Diferente do Redux usual, as nossas funções não retornam apenas um objeto referente a ação, e sim uma função: **dispatch=>{ }**.

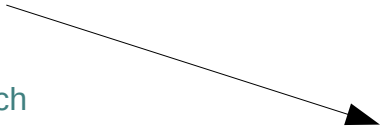
Isso se deve graças a Middleware redux-thunk, pois no caso de signin, signout e signup temos operações **assíncronas**.

Para tratar operações assíncronas, o redux-thunk envolve as ações {type, payload} dentro da função **dispatch=>{ }**, possibilitando assim que sua aplicação não “trave” ao fazer uma **chamada remota ao Firebase**.

authActionCreator.js

signup

```
.catch(
  (error)=>{
    dispatch({
      type: SIGNUP_ERROR,
      payload: {authMessage: `Erro na criação do usuário: ${error}`}
    })
    callback()
  }
)
} catch(error){
  dispatch({
    type: SIGNUP_ERROR,
    payload: { authMessage: `Erro na conexão com o firebase: ${error}`}
  })
  callback()
} //try-catch
} //return dispatch
}
```



A função **callback()**, passada como parâmetro, possibilita aos componentes que chamarem essas ações “esperarem” dentro do callback a computação remota acabar. Isto pode ser interessante em casos que a interface remota queira renderizar uma tela de loading, que só fecha quando o resultado remoto é obtido.

Veremos mais detalhes do uso do callback no signin e signup, na interface.


```
export const signin = (email,password,callback) =>{  
  return dispatch => {  
    try{  
      firebase  
      .auth()  
      .signInWithEmailAndPassword(email,password)  
      .then(  
        (data)=>{  
          dispatch({  
            type: SIGNIN_SUCCESS,  
            payload: {  
              authMessage: `Login efetuado com sucesso`,  
              userMail: data.user.email}  
            })  
          callback()  
        }  
      )  
    }  
    .catch(  
      (error)=>{  
        dispatch({  
          type: SIGNIN_ERROR,  
          payload: {authMessage: `Erro login do usuário: ${error}`}  
        })  
        callback()  
      }  
    )  
  }  
}
```

authActionCreator.js

signin

A lógica do Signin não é muito diferente de Signout.

authActionCreator.js

signin

```
    }catch(error){
      dispatch({
        type: SIGNIN_ERROR,
        payload: { authMessage: `Erro na conexão com o firebase: ${error}` }
      })
      callback()
    }
  }
}
```

authActionCreator.js

signout

```
export const signout = (callback) => {  
  return dispatch => {  
    try{  
      firebase  
        .auth()  
        .signOut()  
        .then(  
          ()=>{  
            dispatch({  
              type: SIGNOUT_SUCCESS,  
              payload: {authMessage: `Signout efetuado com sucesso`}  
            })  
            callback()  
          }  
        )  
      .catch(  
        (error)=>{  
          dispatch({  
            type: SIGNOUT_ERROR,  
            payload: {authMessage: `Erro logout: ${error}`}  
          })  
          callback()  
        }  
      )  
    }  
  }  
}
```

authActionCreator.js

signout

```
    } catch (error) {  
      dispatch({  
        type: SIGNOUT_ERROR,  
        payload: { authMessage: `Erro na conexão com o firebase: ${error}` }  
      })  
      callback()  
    }  
  }  
}
```

Componentes Auxiliares

- Dentro da pasta commons, iremos criar dois componentes de interface auxiliares:
 - Card.jsx → Acesso irrestrito
 - RestrictedCard.jsx → Acessível apenas com o usuário logado

Card.jsx

```
import React, { Component } from 'react'

export default class Card extends Component {
  render() {
    return (
      <div className='content'>
        <div className='card'>
          <div className='card-header'>
            {this.props.title}
          </div>
          <div className='card-body'>
            {this.props.children}
          </div>
        </div>
      </div>
    )
  }
}
```

*Usa os "cards" do bootstrap. Mais detalhes em
<https://getbootstrap.com/docs/4.3/components/card/>*

RestrictedCard.jsx

```
import React, {Component} from 'react'
import Card from './Card'

import { connect } from 'react-redux'

class RestrictedCard extends Component{

  componentDidMount(){
    if(this.props.firebaseAuth.isLoaded && this.props.firebaseAuth.isEmpty){
      this.props.history.push('/signin')
    }
  }

  render(){
    return (
      <Card title={this.props.title}>
        {this.props.children}
      </Card>
    )
  }
}
```

Agora, fazendo uso do redutor do firebase, definido em reducers/index.js, eu Testo se o usuário ainda está logado.

```
function mapStateToProps(state) {
  return {
    firebaseAuth: state.firebaseReducer.auth
  }
}

export default connect(mapStateToProps)(RestrictedCard)
```

Colando o Redux

- Para inserir o Redux, temos que criar os objetos no arquivo principal da aplicação, o **index.js** de **src**.
- Você deve seguir o que diz a documentação:
 - <http://react-redux-firebase.com/docs/v3-migration-guide.html>

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import 'bootstrap/dist/css/bootstrap.min.css'
import App from './App';
import * as serviceWorker from './serviceWorker';

import { createStore, applyMiddleware } from 'redux'
import { Provider } from 'react-redux'
import { ReactReduxFirebaseProvider } from 'react-redux-firebase'
import reduxThunk from 'redux-thunk'

import reducer from './store/reducers' //chamando index.js
import firebase from './utils/firebase'

const store = createStore(
  reducer,
  {},
  applyMiddleware(reduxThunk)
)

const rrfProps = {
  firebase,
  config: {},
  dispatch: store.dispatch
}
```

```
ReactDOM.render(
  <Provider store={store}>
    <ReactReduxFirebaseProvider {...rrfProps}>
      <App />
    </ReactReduxFirebaseProvider>
  </Provider>
  , document.getElementById('root')
);

serviceWorker.unregister();
```

Cria o estado geral e aplica a middleware, que trata de comunicação assíncrona.

Configurações gerais do Firebase, usadas pelo "react-redux-firebase"

Componentes da Aplicação

- Os componentes da aplicação são:
 - **Main.jsx** → Apenas pra encher linguiça, escreva qualquer coisa.
 - **Signinjsx** → Login
 - **Signup.jsx** → Cadastro
 - **Content A e B .jsx** → páginas de acesso restrito
 - **App.jsx** → Página raiz

App.jsx

```
import React, { Component } from 'react'
import { BrowserRouter, Link, Switch, Route } from 'react-router-dom'
import Main from './components/Main'
import Signin from './components/Signin'
import Signup from './components/Signup'
import ContentA from './components/ContentA'
import ContentB from './components/ContentB'
import { connect } from 'react-redux'
```

Importando os componentes.

```
class App extends Component {
  render() {
    return (
      <BrowserRouter>
        <div className='container'>
          <nav className='navbar navbar-expand-lg navbar-light bg-light'>
            <Link to={'/'} className='navbar-brand'>PET Firebase Auth</Link>
            <div className='collapse navbar-collapse' id='navbarSupportedContent'>
              <ul className='navbar-nav mr-auto'>
                <li>
                  <Link to={'/'} className='nav-link'>Home</Link>
                </li>
                <li>
                  <Link to={'/signin'} className='nav-link'>Login</Link>
                </li>
                <li>
                  <Link to={'/signup'} className='nav-link'>Cadastrar</Link>
                </li>
                <li>
                  <Link to={'/contentA'} className='nav-link'>Conteúdo A</Link>
                </li>
                <li>
                  <Link to={'/contentB'} className='nav-link'>Conteúdo B</Link>
                </li>
              </ul>
              {this.props.user}
            </div>
          </nav>
        </div>
      </BrowserRouter>
    )
  }
}
```

Imprimindo o valor da variável do estado geral, "user".

```
</nav>
<Switch>
  <Route exact path='/' component={Main}/>
  <Route path='/signin' component={Signin}/>
  <Route path='/signup' component={Signup}/>
  <Route path='/contentA' component={ContentA}/>
  <Route path='/contentB' component={ContentB}/>
</Switch>
</div>
</BrowserRouter>
)
}
}

function mapStateToProps(state){
  return{
    user: state.authReducer.user
  }
}

export default connect(mapStateToProps)(App)
```

Signup.jsx (Cadastrar)

```
import React, { Component } from 'react'
import Card from './commons/Card'

import { connect } from 'react-redux'
import { signup } from '../store/actions/authActionCreator'
```

```
class Signup extends Component {

  constructor(props) {
    super(props)
    this.state = { login: "", password: "" }
    this.setLogin = this.setLogin.bind(this)
    this.setPassword = this.setPassword.bind(this)
    this.onSubmit = this.onSubmit.bind(this)
    this._isMounted = false
  }

  componentDidMount(){
    this._isMounted = true
  }

  componentWillUnmount(){
    this._isMounted = false
  }

  setLogin(e) {
    this.setState({ login: e.target.value })
  }

  setPassword(e) {
    this.setState({ password: e.target.value })
  }
}
```

Signup.jsx (Cadastrar)

```
onSubmit(e) {
  e.preventDefault()
  this.setState({loading:true})

  this.props.mySignup(this.state.login, this.state.password, ()=>{
    this._isMounted && this.setState({loading:false})
  })

  this.setState({ login: "", password: ""})
}

renderButton() {
  if (this.state.loading) {
    return (
      <button className="btn btn-primary" type="button" disabled>
        <span className="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>
        Carregando...
      </button>
    )
  }

  return (
    <input type='submit' value='Cadastrar' className='btn btn-primary' />
  )
}

renderMessage() {
  if (this.props.userMsg) {
    const msgType = (this.props.userMsg.includes('Err') ? 'alert-danger' : 'alert-info')
    return (
      <div className={`alert ${msgType}`} style={{ marginTop: '10px' }}>
        {this.props.userMsg}
      </div>
    )
  }
  return
}
```

A implementação da função callback só será chamada quando a operação remota terminar.

Signup.jsx (Cadastrar)

```
render() {
  return (
    <Card title='Cadastrar'>
      <form onSubmit={this.onSubmit}>
        <div className='form-group'>
          <label>Login: </label>
          <input type='text' className='form-control'
            value={this.state.login} onChange={this.setLogin} />
        </div>
        <div className='form-group'>
          <label>Password: </label>
          <input type='password' className='form-control'
            value={this.state.password} onChange={this.setPassword} />
        </div>
        {this.renderButton()}
      </form>
      {this.renderMessage()}
    </Card>
  )
}

function mapStateToProps(state) {
  return {
    userMsg: state.authReducer.authMsg
  }
}

function mapDispatchToProps(dispatch) {
  return {
    mySignup(login, password, callback) {
      const action = signup(login, password, callback)
      dispatch(action)
    }
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(Signup)
```

Signin.jsx (Fazer login)

```
import React, {Component} from 'react'
import Card from './commons/Card'

import { connect } from 'react-redux'
import { signin } from '../store/actions/authActionCreator'

class Signin extends Component{

  constructor(props){
    super(props)
    this.state = {login:"",password:"",loading:false}
    this.setLogin = this.setLogin.bind(this)
    this.setPassword = this.setPassword.bind(this)
    this.onSubmit = this.onSubmit.bind(this)
  }

  setLogin(e){
    this.setState({login:e.target.value})
  }

  setPassword(e){
    this.setState({password:e.target.value})
  }

  renderButton() {
    if (this.state.loading) {
      return (
        <button className="btn btn-primary" type="button" disabled>
          <span className="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>
            Carregando...
        </button>
      )
    }

    return (
      <input type='submit' value='Efetuar Login' className='btn btn-primary' />
    )
  }
}
```

Signin.jsx (Fazer login)

```
renderMessage() {  
  if (this.props.userMsg) {  
    const msgType = (this.props.userMsg.includes('Err') ? 'alert-danger' : 'alert-info')  
    return (  
      <div className={`alert ${msgType}`} style={{ marginTop: '10px' }}>  
        {this.props.userMsg}  
      </div>  
    )  
  }  
  return  
}  
  
onSubmit(e){  
  e.preventDefault()  
  this.setState({loading:true})  
  
  this.props.mySignin(this.state.login,this.state.password, ()=>{  
    this.setState({loading:false})  
  })  
  
  this.setState({login:"",password:""})  
}
```


Signin.jsx (Fazer login)

```
render(){
  return (
    <Card title='Fazer Login'>
      <form onSubmit={this.onSubmit}>
        <div className='form-group'>
          <label>Login: </label>
          <input type='text' className='form-control'
            value={this.state.login} onChange={this.setLogin} />
        </div>
        <div className='form-group'>
          <label>Password: </label>
          <input type='password' className='form-control'
            value={this.state.password} onChange={this.setPassword} />
        </div>
        {this.renderButton()}
      </form>
      {this.renderMessage()}
    </Card>
  )
}

function mapStateToProps(state) {
  return {
    userMsg: state.authReducer.authMsg
  }
}

function mapDispatchToProps(dispatch){
  return{
    mySignin(login,password,callback) {
      const action = signin(login,password,callback)
      dispatch(action)
    }
  }
}

export default connect(mapStateToProps,mapDispatchToProps)(Signin)
```

ContentA e B

```
import React, {Component} from 'react'
import Card from './commons/RestrictCard' Usando o Card restrito.
```

```
import { connect } from 'react-redux'
import { signout } from '../store/actions/authActionCreator'
```

```
class ContentA extends Component{

  logout(){
    this.props.mySignout(
      ()=>{
        this.props.history.push('/signin')
      }
    )
  }

  render(){
    return (
      <Card title='Conteúdo A' history={this.props.history}>
        Conteúdo apenas para usuários. <br /><br />
        <button className='btn btn-danger'
          onClick={()=>this.logout()}
        >
          Fazer Logout
        </button>
      </Card>
    )
  }
}
```

ContentA e B

```
function mapStateToProps(state) {  
  return {  
    userMsg: state.authReducer.authMsg,  
    firebaseAuth: state.firebaseReducer.auth  
  }  
}  
  
function mapDispatchToProps(dispatch){  
  return{  
    mySignout(callback) {  
      const action = signout(callback)  
      dispatch(action)  
    }  
  }  
}  
  
export default connect(mapStateToProps,mapDispatchToProps)(ContentA)
```

Referências

- Código dessa aula foi baseado no projeto de:
 - <https://github.com/clairechabas/firebase-auth-react-redux>

Ajustes na Aplicação de Login

Introdução

- De posse da aplicação passada, iremos fazer as seguintes alterações:
 - Manter estado durante o **refresh** da página;
 - Envio de e-mail para confirmação.

Manter o Estado

- Para manter o estado geral da aplicação, temos duas soluções:
 - Usar uma middleware dedicada (redux-persist)
 - Usar o LocalStorage ou SessionStorage
- Para essa aula, usaremos a abordagem mais simples, o LocalStorage.

storeConfig.js

- Passaremos as configurações de index.js (raiz) para um arquivo a parte.
- Na pasta store crie o arquivo storeConfig.js (alguns projetos gostam de chamar apenas de store.js)

storeConfig.js

```
import { createStore, applyMiddleware } from 'redux'  
import reduxThunk from 'redux-thunk'
```

```
import reducer from '../store/reducers'  
import firebase from '../utils/firebase'
```

```
const persistedState = loadFromLocalStorage()
```

```
const store = createStore(  
  reducer,  
  persistedState,  
  applyMiddleware(reduxThunk)  
)
```

```
const rrfProps = {  
  firebase,  
  config: {},  
  dispatch: store.dispatch  
}
```

```
export { store, rrfProps }
```

index.js(raiz)

```
import { Provider } from 'react-redux'  
import { ReactReduxFirebaseProvider } from 'react-redux-firebase'
```

```
import { store, rrfProps } from './store/storeConfig'
```

```
ReactDOM.render(  
  <Provider store={store}>  
    <ReactReduxFirebaseProvider {...rrfProps}>  
      <App />  
    </ReactReduxFirebaseProvider>  
  </Provider>  
  , document.getElementById('root')  
);
```

storeConfig.js

- Voltando ao storeConfig, você deverá adicionar funções para salvar o estado geral e também carregá-lo.
- O salvamento do estado geral será feito toda vez que o mesmo é modificado.
- O carregamento do estado geral será feito durante a criação só estado geral (store), através do método createStore, chamado quando a aplicação é carregada ou recarregada.

storeConfig.js

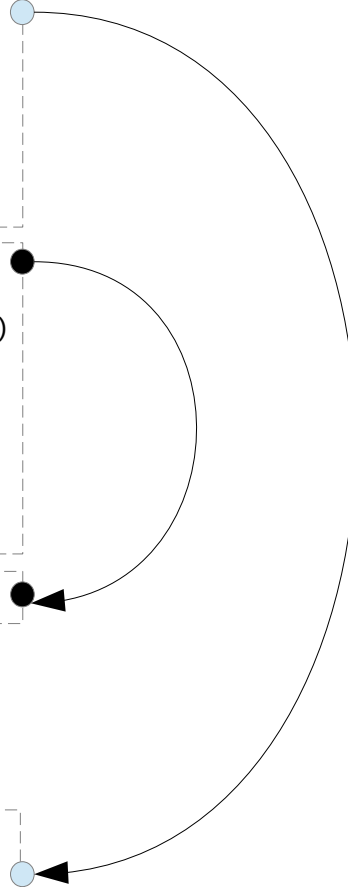
```
....  
function saveToLocalStorage(state) {  
  try {  
    const serializedState = JSON.stringify(state)  
    localStorage.setItem('state', serializedState)  
  } catch (error) {  
    console.log(error)  
  }  
}
```

```
function loadFromLocalStorage() {  
  try {  
    const serializedState = localStorage.getItem('state')  
    if (serializedState === null) return undefined  
    return JSON.parse(serializedState)  
  } catch (error) {  
    console.log(error)  
    return undefined  
  }  
}
```

```
const persistedState = loadFromLocalStorage()
```

```
const store = createStore(  
  reducer,  
  persistedState,  
  applyMiddleware(reduxThunk)  
)
```

```
store.subscribe(  
  () => {  
    saveToLocalStorage(store.getState())  
  }  
)  
...  
...  
...
```



Verificação de E-mail

- Outra feature interessante do Firebase é que ele permite a verificação de e-mail.
- Ele envia um e-mail no momento do **signup** e, durante o **signin** você deve verificar se o usuário verificou o e-mail.
- Nas páginas de content você só deixa o usuário acessar caso ele tenha verificado o e-mail (exercício ded redux).

signup

```
export const signup = (email, password, callback) => {  
  return dispatch => {  
    try {  
      firebase  
        .auth()  
        .createUserWithEmailAndPassword(email, password)  
        .then(  
          () => {  
            firebase.auth().onAuthStateChanged(  
              (user) => {  
                user.sendEmailVerification();  
              });  
          })  
        .then(  
          () => {  
            firebase.auth().onAuthStateChanged(  
              (user) => {  
                if (user) {  
                  dispatch({  
                    type: SIGNUP_SUCCESS,  
                    payload: {  
                      authMessage: `Cadastro efetuado com sucesso! Verifique seu e-mail`,  
                      userMail: "",  
                      verified: false  
                    }  
                  })  
                  .callback()  
                } else {  
                  dispatch({  
                    type: SIGNUP_ERROR,  
                    payload: { authMessage: `Não foi possível conectar` }  
                  })  
                  .callback()  
                }  
              })  
            }  
          )  
        )  
      //firebase.auth().onAuthStateChanged(  
    }  
  }  
  //se deu certo  
}
```

signup

```
.catch(  
  (error) => {  
    dispatch({  
      type: SIGNUP_ERROR,  
      payload: { authMessage: `Erro na criação do usuário: ${error}` }  
    })  
    callback()  
  }  
)  
} catch (error) {  
  dispatch({  
    type: SIGNUP_ERROR,  
    payload: { authMessage: `Erro na conexão com o firebase: ${error}` }  
  })  
  callback()  
} //try-catch  
  
} //return dispatch  
}
```

signin

```
...
.signinWithEmailAndPassword(email, password)
  .then(
    (data) => {

      if(!data.user.emailVerified){
        dispatch({
          type: EMAIL_NOT_VERIFIED,
          payload: {
            authMessage: `E-mail não verificado. Veja sua caixa de e-mail`,
            verified: false
          }
        })
      }else{
        dispatch({
          type: SIGNIN_SUCCESS,
          payload: {
            authMessage: `Login efetuado com sucesso`,
            userMail: data.user.email,
            verified: true
          }
        })
      }
    }
  )
  .callback()
)
...

```


authReducer.js

```
const INITIAL_STATE = {
  authMsg: null,
  user: "",
  verified: false
}

export default function (state = INITIAL_STATE, action) {
  switch (action.type) {
    case SIGNUP_SUCCESS:
      return {
        ...state,
        authMsg: action.payload.authMessage,
        verified: action.payload.verified
      }
    case SIGNUP_ERROR:
      return {
        ...state,
        authMsg: action.payload.authMessage
      }
    case SIGNIN_SUCCESS:
      return {
        ...state,
        authMsg: action.payload.authMessage,
        user: action.payload.userMail,
        verified: action.payload.verified
      }
    case SIGNIN_ERROR:
      return {
        ...state,
        authMsg: action.payload.authMessage
      }
    case SIGNOUT_SUCCESS:
      return {
        user: null,
        authMsg: action.payload.authMessage,
        verified: action.payload.verified
      }
    case SIGNOUT_ERROR:
      return {
        ...state,
        authMsg: action.payload.authMessage
      }
    case RESET_AUTH_MESSAGE:
      return {
        ...state,
        authMsg: null
      }
    case EMAIL_NOT_VERIFIED:
      return {
        ...state,
        authMsg: action.payload.authMessage,
        verified: action.payload.verified
      }
    default:
      return state
  }
}
```

RestrictedCard.jsx

```
class RestrictedCard extends Component{

  componentDidMount(){

    if(this.props.firebaseAuth.isLoaded && this.props.firebaseAuth.isEmpty){
      this.props.history.push('/signin')
    }

    if(!this.props.emailVerified){
      this.props.history.push('/signin')
    }
  }

  render(){
    return (
      <Card title={this.props.title}>
        {this.props.children}
      </Card>
    )
  }
}

function mapStateToProps(state) {
  return {
    firebaseAuth: state.firebaseReducer.auth,
    emailVerified: state.authReducer.verified
  }
}
```