

React Native

Projeto de Interfaces de Dispositivos Móveis

**Projeto Autenticador com
Firebase**

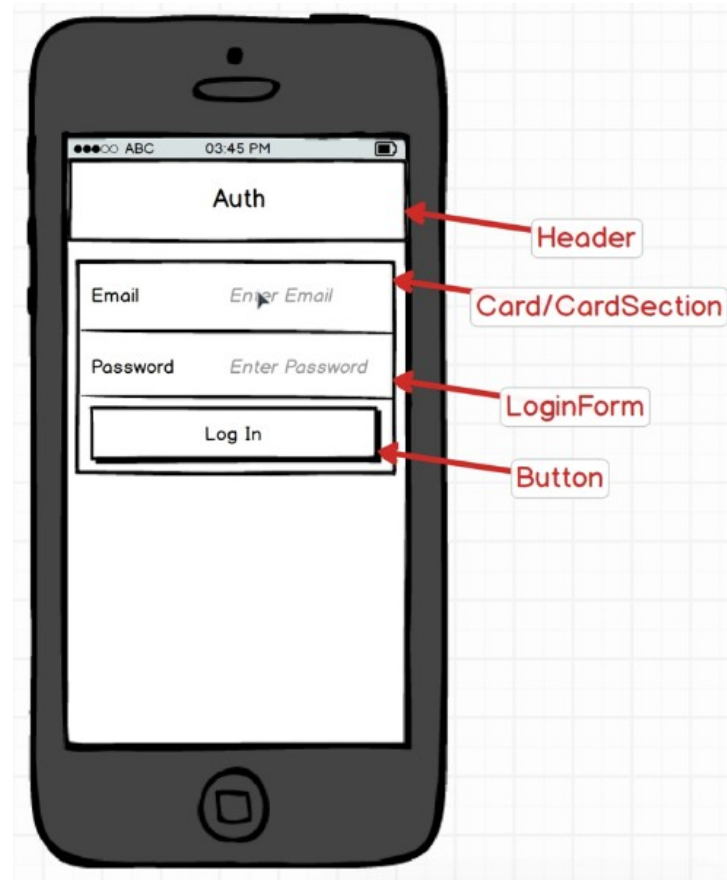
Aula 05

Introdução

- **Projeto Albums - Parte 2**
 - Construção de uma nova aplicação
 - Reuso de componentes da aplicação passada
 - Autenticação com o Firebase

Introdução

- Visão Geral:



Introdução

- Crie um novo projeto:
 - `react-native init projeto_auth`
 - Reescreva o arquivo `App.js`
 - Crie a pasta “src”, no diretório raiz e coloque `App.js` lá
 - Mude o arquivo `index.js` para apontar para a nova localização de `App.js`

App.js (dentro de /src)

```
import React,{Component} from 'react';
import {View,Text} from 'react-native';

export default class App extends Component{
  render(){
    return(
      <View style={{flex:1,justifyContent:"center",alignItems:"center"}}>
        <Text>Aplicação de Autorização</Text>
      </View>
    );
  }
}
```

Reusando os Componentes

- Em src:
 - crie a pasta /components/common
 - Em /common, jogue os componentes do projeto anterior:
 - Header.js, MyButton.js, Card.js, CardItem.js
 - Crie o arquivo “index.js” em /common

Reusando os Componentes

- /common/index.js

```
export * from './MyButton';  
export * from './Card';  
export * from './CardItem';  
export * from './Header';
```

Reusando os Componentes

- Em cada componente dentro de /common, modifique (exemplo):

```
//...  
class Header extends Component {  
//...  
export {Header}
```

Só é necessário fazer essa modificação por causa do arquivo index.js!

Reusando os Componentes

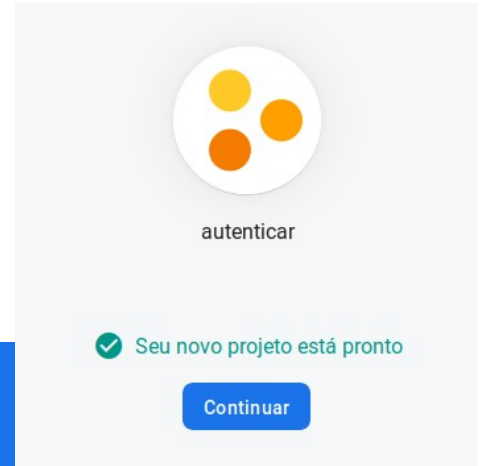
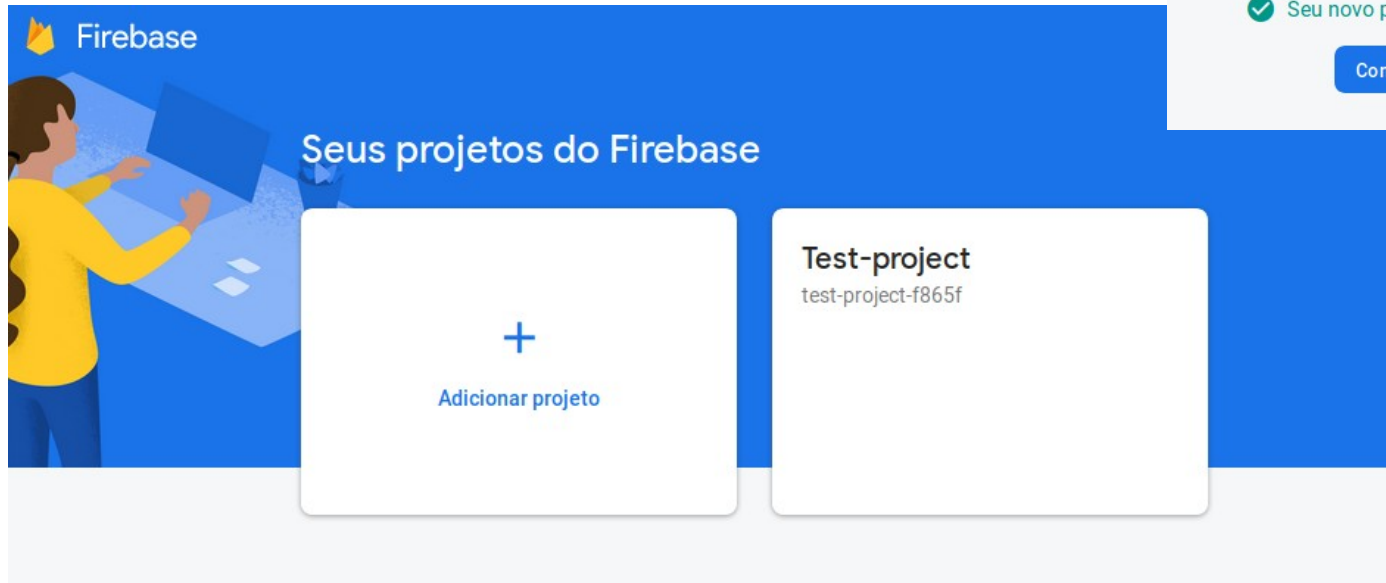
- Em App.js

```
//...
import {Header} from './components/common'

export default class App extends Component{
  render(){
    return(
      <View style={{flex:1,justifyContent:"center",alignItems:"center"}}>
        <Header title="Aula Parte 2"/>
        <Text>Aplicação de Autorização</Text>
      </View>
    );
  }
}
```

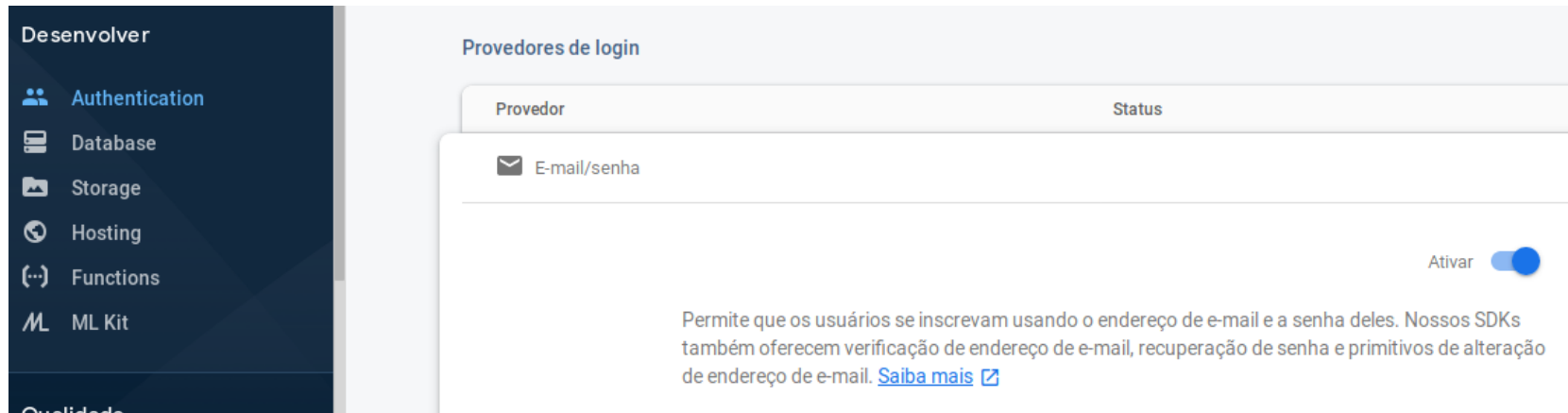
Firebase

- <https://firebase.google.com>



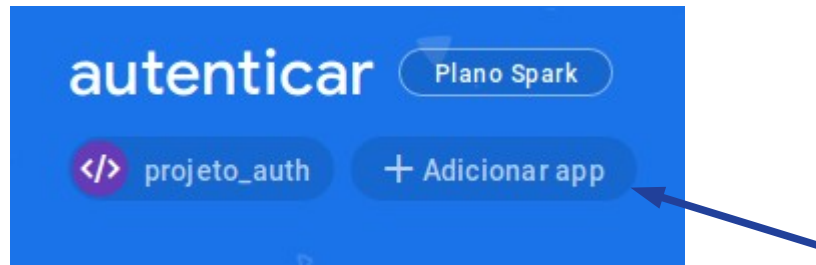
Firebase

- Crie um novo projeto e ative autenticação por login e senha.



Firestore

- Instale o Firestore no seu projeto react-native:
 - `npm install firestore --save`
- No console do Firestore, registre uma nova aplicação:



Firebase

- Copie o objeto referente à conexão:

```
<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyCOWJ3XKtV6VE74VLE05NmDVFCcXTTDnVSIk",
    authDomain: "autenticar-600a4.firebaseio.com",
    databaseURL: "https://autenticar-600a4.firebaseio.com",
    projectId: "autenticar-600a4",
    storageBucket: "",
    messagingSenderId: "719799626205",
    appId: "1:719799626205:web:57281285dd3fa8bfb26e34",
    measurementId: "G-EKV411S82V"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
  firebase.analytics();
</script>
```

Firebase

- Em App.js

```
//...
import firebase from 'firebase';

import {Header} from './components/common'

export default class App extends Component{

  componentDidMount(){
    if(!firebase.apps.length){
      firebase.initializeApp({
        apiKey: 'AIzaSyCOWJXKYoSDDVLoC5NwDVFCtXTTDnV3Ik',
        authDomain: "autenticar-758a4.firebaseio.com",
        databaseURL: "https://autenticar-77ds7a4.firebaseio.com",
        projectId: "autenticar-633sd0a4",
        storageBucket: "",
        messagingSenderId: "7197dssdds$554299626205",
        appId: "1:719799626205:web:57281285sddf453edfrfr43345efg",
        measurementId: "G-EKV411S$%RE#382V"
      });
    }
  }
  //...
```

LoginForm

- LoginForm.js

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

export default class LoginForm extends Component{
  render(){
    return(
      <Card>
        <CardItem/>
        <CardItem/>
        <CardItem>
          <MyButton>
            Log in
          </MyButton>
        </CardItem>
      </Card>
    );
  }
}
```

LoginForm

- App.js

```
import LoginForm from './components/LoginForm';

export default class App extends Component{

  componentDidMount(){
    //...
  }

  render(){
    return(
      <View>
        <Header title="Autenticação"/>
        <LoginForm/>
      </View>
    );
  }
}
```


LoginForm

- Criando os TextInputs
(versão não reusável)

```
import React, { Component } from 'react';
import { TextInput } from 'react-native';
import { Card, CardItem, MyButton } from './commons'

export default class LoginForm extends Component {
  constructor(props) {
    super(props);
    this.state = { email: "", senha: "" }
  }
  render() {
    return (
      <Card>
        <CardItem>
          <TextInput
            style={{fontSize:16}}
            placeholder='Entre com seu e-mail'
            onChangeText={email => this.setState({ email })}
          />
        </CardItem>

        <CardItem>
          <TextInput
            style={{fontSize:16}}
            placeholder='Entre com sua senha'
            onChangeText={senha => this.setState({ senha })}
          />
        </CardItem>

        <CardItem>
          <MyButton>
            Log in
          </MyButton>
        </CardItem>
      </Card>
    );
  }
}
```

MyInput (TextInput Reusável)

- Na pasta commons,
crie o arquivo:
 - MyInput.js

```
import React,{Component} from 'react';
import {View,Text,TextInput} from 'react-native';

class MyInput extends Component{
  render(){
    return(
      <View>
        <Text>{this.props.label}</Text>
        <TextInput
          style={{fontSize:16}}
          placeholder={this.props.placeholder}
          onChangeText={this.props.onChangeText}
        />
      </View>
    );
  }
}

export {MyInput}
```

CUIDADO



MyInput (TextInput Reusável)

- Arquivo /commons/index.js

```
export * from './MyButton';  
export * from './Card';  
export * from './CardItem';  
export * from './Header';  
export * from './MyInput';
```

MyInput

(TextInput Reusável)

```
//...
import { Card, CardItem, MyButton, MyInput } from './commons'

export default class LoginForm extends Component {
  constructor(props) {
    super(props);
    this.state = { email: "", senha: "" }
  }
  render() {
    return (
      <Card>
        <CardItem>
          <MyInput
            label = 'E-mail'
            placeholder='Entre com seu e-mail'
            onChangeText={email => this.setState({ email })}
          />
        </CardItem>

        <CardItem>
          <MyInput
            label = 'Senha'
            placeholder='Entre com sua senha'
            onChangeText={senha => this.setState({ senha })}
          />
        </CardItem>
      </Card>
    )
  }
}
```

MyInput (Estilos)

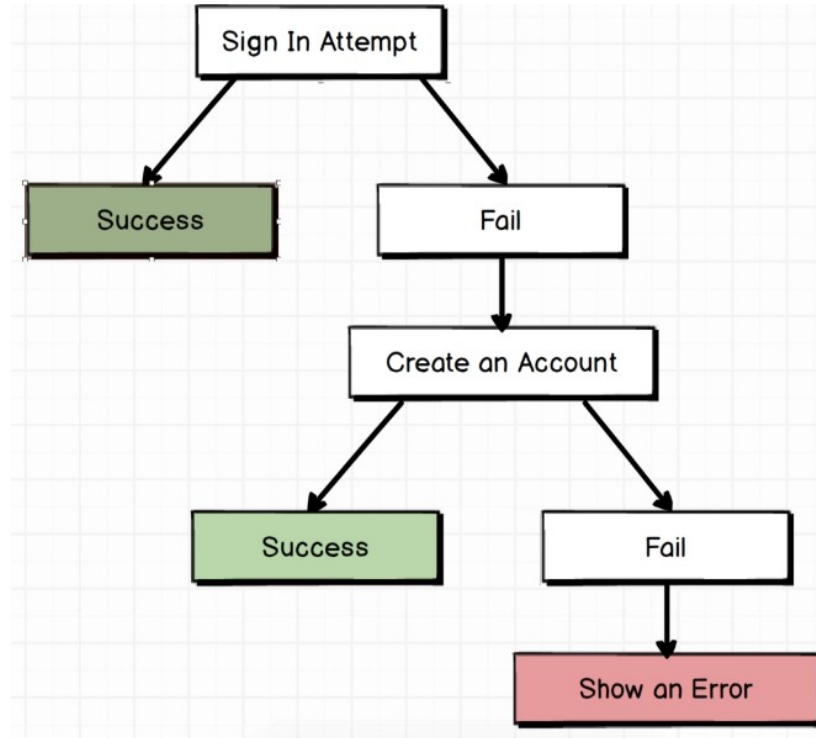
```
import React,{Component} from 'react';
import {View,Text,TextInput, StyleSheet} from 'react-native';

class MyInput extends Component{
  render(){
    return(
      <View style={estilos.containerEstilo}>
        <Text style={estilos.labelEstilo}>{this.props.label}</Text>
        <TextInput
          style={estilos.inputEstilo}
          placeholder={this.props.placeholder}
          onChangeText={this.props.onChangeText}
        />
      </View>
    );
  }
}
```

```
const estilos = StyleSheet.create({
  containerEstilo:{
    flex:1,
    flexDirection:"row",
    alignItems:"center",
    height:40
  },
  labelEstilo:{
    fontSize:18,
    paddingLeft:10,
    flex:1
  },
  inputEstilo:{
    color:'#000',
    paddingRight:5,
    paddingLeft:5,
    fontSize:18,
    lineHeight:23,
    flex:4
  }
});
```

Processando as credencias no Firebase

Fluxo Lógico



LoginForm.js

```
constructor(props) {  
  super(props);  
  this.state = { email: "", senha: "", error: "" }  
}  
  
acaoBotao(){  
  //alert(this.state.email);  
  firebase.auth().signInWithEmailAndPassword(this.state.email,this.state.senha)  
  .catch(()=>{ //problema no e-mail e na senha  
    firebase.auth().createUserWithEmailAndPassword(this.state.email,this.state.senha)  
    .catch((error)=>{ //algum problema na criação do usuário  
      this.setState({error:error.message})  
    });  
  });  
}  
//...  
  
</CardItem>  
<Text style={{fontSize:20,color:'red',alignSelf:"center"}}>  
  {this.state.error}  
</Text>  
<CardItem>  
  <MyButton action={this.acaoBotao.bind(this)}>  
    Log in
```


Console Firebase

PIDM-2019-2 ▾ Ir para a documentação 🔔

Authentication

[Usuários](#) [Método de login](#) [Modelos](#) [Uso](#)

[Adicionar usuário](#) ↺ ⋮

Identificador	Provedores	Criado em	Conectado	UID do usuário ↑
jeff@gmail.com	✉	17 de out de 2...	17 de out de 2...	GDsSAsnTQtaWS3LI3zVbMtP2c8L2

Linhas por página: 50 ▾ 1-1 de 1 < >

MySpinner.js (loading...)

```
import React, {Component} from 'react';
import {View, ActivityIndicator} from 'react-native';

class MySpinner extends Component{
  render(){
    <View style={{flex:1,justifyContent:"center",alignItems:"center"}}>
      <ActivityIndicator size={this.props.size || 'large'}/>
    </View>
  }
}

export {MySpinner}
```

Loading...

- Ao clicar no botão, o spinner deve aparecer em seu lugar.
- Caso algum erro ocorra, o texto de erro é mostrado e o botão aparece novamente, para uma nova tentativa.

LoginForm.js

- Criar uma variável **loading** em state, com estado inicial “**false**”.
- Mudar o valor para “**true**” quando pressionar o botão de login.
- Solução: criar uma função que retorne o JSX adequado (renderBotao)

LoginForm.js

```
constructor(props) {  
  super(props);  
  this.state = { email: "", senha: "", error: "", loading: false }  
}  
//...  
renderBotao() {  
  if (this.state.loading) {  
    return <MySpinner size='small' />;  
  }  
  return (  
    <MyButton action={this.acaoBotao.bind(this)}>  
      Log in  
    </MyButton>  
  );  
}  
//...
```

Esse código deve ser colocado no lugar do botão, no render principal de LoginForm.js.

Veja no próximo slide!

LoginForm.js

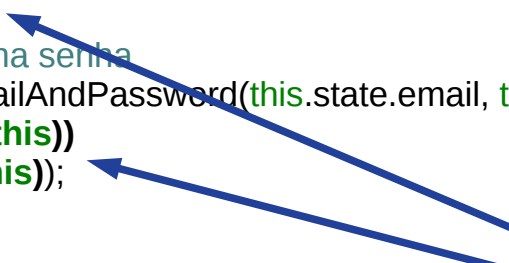
```
<CardItem>
  <MyInput
    label='Senha'
    placeholder='Entre com sua senha'
    secureTextEntry={true}
    onChangeText={senha => this.setState({ senha })}
  />
</CardItem>
<Text style={{ fontSize: 20, color: 'red', alignSelf: "center" }}>
  {this.state.error}
</Text>
<CardItem>
  {this.renderBotao()}
</CardItem>
```

Função Sucesso e Falha

- **Função de sucesso:** caso o login tenha ocorrido ok.
- **Função de falha:** o contrário. O usuário continua na tela de login.

LoginForm.js

```
acaoBotao() {  
  this.setState({ error: "", loading: true });  
  //alert(this.state.email);  
  firebase.auth().signInWithEmailAndPassword(this.state.email, this.state.senha)  
    .then(this.loginSucesso.bind(this))  
    .catch(() => { //problema no e-mail e na senha  
      firebase.auth().createUserWithEmailAndPassword(this.state.email, this.state.senha)  
        .then(this.loginSucesso.bind(this))  
        .catch(this.loginFalhou.bind(this));  
    });  
}
```



Por ser um promessa, use o **this** para manter o contexto!

```
loginSucesso(){  
  this.setState({ email: "", senha: "", error: "", loading: false });  
}
```

```
loginFalhou(error){  
  this.setState({ error: error.message })  
}
```


Logged in e Logged out

- Queremos mostrar diferentes telas quando o usuário:
 - Está logado (logged in)
 - Está “deslogado” (logged out)

Logged in e Logged out

- A solução mais simples para esse problema, e que envolve menos código, é usar das **funcionalidades** do **firebase**. Mais especificamente:
 - `firebase.auth().onAuthStateChanged((user)=>{
 //CÓDIGO da lógica quando o estado da autenticação muda!
})`

Logged in e Logged out

- **onAuthStateChanged** é uma função do serviço de autenticação do firebase que trata eventos relacionados ao login. Ela recebe uma função como entrada, cujo parâmetro “user” identifica:
 - um objeto usuário, caso o login tenha sido um sucesso.
 - mostraremos uma tela de logout
 - null ou undefined, caso contrário.
 - mostraremos uma tela de login.
- Criar a variável **loggedIn**

App.js

```
constructor(props){  
  super(props);  
  this.state = {loggedIn:false};  
}
```

```
componentDidMount() {
```

```
  //chave aqui  
  if (!firebase.apps.length) {  
    //...  
  }
```

```
  firebase.auth().onAuthStateChanged((user)=>{  
    if(user){  
      this.setState({loggedIn:true});  
      //alert("SUCESSO!")  
    }else{  
      this.setState({loggedIn:false});  
      //alert("FALHOU")  
    }  
  })  
}
```

```
renderConteudo(){  
  switch(this.state.loggedIn){  
    case true:  
      return  
        <Card>  
          <CardItem>  
            <MyButton>  
              Log Out  
            </MyButton>  
          </CardItem>  
        </Card>;  
    case false:  
      return <LoginForm/>;  
  }  
}  
  
render() {  
  return (  
    <View>  
      <Header title="Autenticação" />  
      {this.renderConteudo()}  
    </View>  
  );  
}
```

Problema de IHC...

- O Firebase é um serviço na web, e “demora” pra responder (deu certo ou não).
- No entanto, se o usuário está logado, dependendo da conexão, ele verá a tela de login por poucos segundos antes de ser direcionado para a tela de logout!
- Seria interessante um “spinner” aparecer, enquanto isso ocorre.

3 estados de loggedIn

- **True:** realmente está logado graças a autenticação do firebase;
- **False:** não está logado, ou seja, o login falhou ou o usuário clicou em logout;
- **null:** o firebase ainda está carregando pra saber se está logado ou não.

3 estados de loggedIn

```
constructor(props){  
  super(props);  
  this.state = {loggedIn:null};  
}  
  
renderConteudo(){  
  switch(this.state.loggedIn){  
    case true:  
      return <Card><CardItem><MyButton>Log Out</MyButton></CardItem></Card>;  
    case false:  
      return <LoginForm/>;  
    default:  
      return <View style={{height:100,justifyContent:"center",alignItems:"center"}}><MySpinner size="large"/></View>  
  }  
}  
  
render() {  
  return (  
    <View>  
      <Header title="Autenticação" />  
      {this.renderConteudo()}  
    </View>  
  );  
}
```

← Inicie o loggedIn com null!

Log Out...

```
renderConteudo(){  
  switch(this.state.loggedIn){  
    case true:  
      return <Card><CardItem><MyButton  
action={()=>firebase.auth().signOut()}>Log  
Out</MyButton></CardItem></Card>;  
    case false:  
      return <LoginForm/>;  
    default:  
      return <MySpinner size="large"/>  
  }  
}
```