

React JS

Uma biblioteca JavaScript para interfaces de usuário

Back-end

- A arquitetura cliente-servidor na web pressupõe dois agentes:
 - O chamado client-side envolve tudo o que é processado pelo navegador.
 - Já o server-side faz referência ao que é processado no servidor
- Apesar de diferentes, os dois lados precisam trabalhar juntos para garantir a usabilidade do site e uma navegação amigável.
- Nesse contexto, a programação back-end está associada ao server-side

Back-end

- O desenvolvimento back-end cuida das engrenagens de uma aplicação web, criando códigos para que as funções do site (ex.: manipulação de banco de dados, acesso a serviços, etc.) sejam executadas. Como o próprio nome sugere, é um trabalho de bastidores.
- Linguagens muito usadas: C, C++, PHP, Python, Java, .Net, Node.js, entre outras. (Não exclusivas)

Front-end

- a programação front-end está associada ao client-side. O front-end é onde encontramos a "cara" de um site ou aplicativo, com design, interface de navegação e ferramentas de interação com o usuário, como áreas de buscas e formulários.

Front-end

- O front-end também inclui elementos que determinam a identidade visual de um site ou aplicativo, por isso, além do conhecimento de linguagens de programação específicas, um desenvolvedor dessa área só tem a ganhar se tiver noções de design.
- Tecnologias front-end: React, XHTML, HTML5, CSS, JavaScript, AJAX, jQuery, CFML, AngularJS, Angular, Vue

React Js

- [Site Oficial](#)
- Foco na criação de componentes de front-end
- Ex: Formulários, Menus, Listas, etc.
- Podem ser reusados em páginas diferentes.

React Js

- Aplicações React focadas no Front-End e misturam: HTML + CSS + JS
- Fazem mais que o JS puro consegue fazer sozinho
- Utilizado para construção de projetos SPA's
- O que são SPA's?
 - Tradicionalmente, em uma aplicação web renderizada pelo servidor, nós possivelmente teríamos um modelo MVC, e cada View deste modelo seria responsável por criar uma tela. Um exemplo clássico de uma aplicação web.

React Js

- Uma SPA é uma forma diferente de trabalhar com uma aplicação.
- Com essa tecnologia, temos apenas uma página principal e todas as outras serão escritas e modificadas pelo JavaScript.
- Em React, normalmente o Back-End ao invés de retornar arquivos/conteúdo HTML formatado, retorna os dados em um formato específico (ex.: JSON).
- E desta forma, o Back-End fica totalmente separado do Front-End.

Instalação e Ambiente

1. Vamos usar o VisualStudioCode
 - a. Para baixar: <https://code.visualstudio.com/>
 - b. Clicar em extensões no menu lateral esquerdo
 - c. Recomendo instalar as extensões:
 - i. "React-Native/React/Redux snippets for es6/es7"
 - ii. "React Js code snippets"
 - iii. "vscode-icons"

Instalação e Ambiente

1. Instalar o Node.js
 - a. Acessar site <https://nodejs.org/en/>
 - b. Escolher a versão LTS
2. Instalar o react globalmente
 - a. Abrir terminal e digite **npm install -g create-react-app**

Criando um projeto

1. Escolher ou criar um diretório
2. No terminal, vá ao diretório criado e digite:
 - a. `create-react-app <nome do seu projeto>`
 - i. ex: **create-react-app meuprojeto**
3. É só abrir a pasta no VSCode
4. Para rodar o projeto:
 - a. no terminal entre na pasta criada (no nosso exemplo: **cd meuprojeto**)
 - b. rode o comando: **npm start**
5. O navegador padrão vai abrir aplicação já na página inicial do seu projeto

Primeiro projeto

- Na pasta "public" fica a página index.html. Ela será a única página do nosso projeto.
- Lembre que estamos fazendo um SPA.
- Nessa página existe um objeto `<div id="root"></div>` e é nele que vamos incluir os nossos componentes.
- A inclusão não será feita via manipulação direta do index.html...faremos via React.

Primeiro projeto

- Quando criamos um projeto React, o framework cria automaticamente alguns arquivos de modelo.
- Aqui, nós não vamos usar esses arquivos. Então, na pasta **src** apague:
 - App.css
 - App.js
 - App.test.js
 - logo.svg
 - setupTests.js

Primeiro Componente

- Agora vamos criar o nosso primeiro componente. Ele vai se chamar "Social".
- O componente social consiste em exibir o texto `<h1>Olá </h1>`
- Na pasta **src**, crie uma pasta chamada **componentes** e nela outra chamada **Social**
- Na pasta Social, crie um arquivo chamado index.js. Esse arquivo vai guardar o código fonte do nosso componente.

Primeiro Componente

```
import React from 'react';
```

```
function Social(){
```

```
  return <h1>Olá </h1>
```

```
}
```

```
export default Social;
```

Primeiro Componente

- Agora vamos incluir o componente <Social/> no index.html.
- Primeiro passo é editar o **/src/index.js**.
- Apague o **import App from './App';**
- Incluir **import Social from './componentes/Social'**
- Depois, edite o código para substituir o componente <App/> pelo <Social/>

```
ReactDOM.render(  
  <React.StrictMode>  
    <App /> <Social />  
  </React.StrictMode>,  
  document.getElementById('root')
```

```
);
```


Function Based Components

- A maneira mais simples de definir um componente é escrever uma função JavaScript:

```
function Social(){  
  return <h1>Olá </h1>  
}
```

- O uso desse componente se dá na forma de tag HTML: <Social/>

Function Based Components

- Componente podem ter propriedades:
 - `<Social nome="Manoel" idade="43"/>`
- As propriedades podem ser passadas por parâmetro na função usando a palavra reservadas **props**.

```
function Social(props){  
  return <h1>Olá {props.nome}. Você tem {props.idade}</h1>  
}
```

Function Based Components

- Podem existir diversos arranjos entre componentes (agregações, associações, etc.). Por exemplo:

```
function Sobre(props){  
  return (<h1>Olá {props.nome}. Você tem {props.idade}</h1>);  
}
```

```
function Social(props){  
  return (  
    <div>  
      <Sobre nome={props.nome} idade={props.idade}/>  
      <h1>Facebook {props.facebook}</h1>  
      <h1>Instagram {props.instagram}</h1>  
    </div>  
  );  
}
```

Function Based Components

```
ReactDOM.render(  
  <React.StrictMode>  
    <Social nome="Manoel" idade="43" facebook="www.meuface" instagram="@manoelnetom"/>  
    <Social nome="Maria" idade="23" facebook="www.meuface" instagram="@Maria"/>  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

Class Based Components

- A opção para criação de componentes baseados em função é o componente a partir de uma classe (Class Based Components)
- Para o nosso exemplo vamos criar um novo projeto chamado de **classcomponents**. Vamos apagar todos os arquivos (igual ao projeto anterior).
- Em src, crie uma pasta components e dentro dela outra chamada App. Na pasta App crie um index.js
- Ajuste o src/index.js para importar o componente App da pasta correta (./components/App).

Class Based Components

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

class App extends Component{

  render(){

    return(

      <div>

        <h1>Conheça a nossa Equipe:</h1>

      </div>

    );

  }

}

export default App;
```

Class Based Components

- Vamos criar um outro componente de classe chamado de <Equipe> e vamos usá-lo no componente <App>.
- Crie /src/components/Equipe/index.js

Class Based Components

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

class Equipe extends Component{

  render(){

    return(

      <div>

        <h1>Nome: {this.props.nome}    </h1>

      </div>

    );

  }

}

export default App;
```


Class Based Components

- O **props** não precisa ser passado por parâmetro. Basta usar **this.props** ...no exemplo **this.props.nome**

Class Based Components

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

class App extends Component{

  render(){

    return(

      <div>

        <h1>Conheça a nossa Equipe:</h1>

        <Equipe nome="Manoel"/>

      </div>

    );

  }

}

export default App;
```

Stateless Components

- Exibem dados estáticos. Não se pode mudar o valor de uma propriedade passada.

```
ReactDOM.render(  
  <React.StrictMode>  
    <Social nome="Manoel" idade="43"/>  
    <Social nome="Maria" idade="23"/>  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

Stateful Components

- São dinâmicos. Podem mudar dinamicamente.
 - Estados (states) são atributos/propriedades mutáveis/dinâmicas de um componente.
-
- Suponha que queremos alterar a propriedade contador de um componente ao clicar em botões.
 - Primeiro passo é definir a variável "contador" como um state de um componente. Isso é feito dentro do construtor de um class component.

Trabalhando com states

```
class App extends Component{  
  constructor(props){  
    super(props);  
    this.state={  
      contador:o  
    }  
  }  
  render(){ .....}
```

Trabalhando com States

- Mudamos o JSX para incluir os botões e configurar o evento onClick.
- No botão "+" chamamos a função aumentar e no "-" a diminuir.
- É preciso fazer o bind das funções no construtor;
- Ou pode-se usar arrow function na função render().

Trabalhando com states (funções e setState)

```
aumentar(){  
    let state=this.state;  
  
    state.contador+=1;  
  
    this.setState(state);  
}
```

Trabalhando com states (funções e setState)

```
diminuir(){  
    let state=this.state;  
  
    state.contador-=1;  
  
    this.setState(state);  
}
```


Trabalhando com states (bind)

```
class App extends Component{  
  constructor(props){  
    super(props);  
    this.state={  
      contador:0  
    }  
    this.aumentar = this.aumentar.bind(this);  
    //this.diminuir = this.diminuir.bind(this);  
  }  
}
```

Trabalhando com states (bind)

```
render(){  
  return (  
    <div>  
      <h1>Contador</h1>  
      <h2><button onClick={this.diminuir}>-</button> {this.state.contador} <button  
onClick={()=> this.aumentar()}>+</button></h2>  
      <button onClick={this.zerar.bind(this)}>Zerar</button>  
    </div>  
  );  
}
```

Ciclos de vida dos Componentes

- Inicialização
- Montagem
- Atualização
- Desmontagem

Ciclos de vida dos Componentes

- **Inicialização:** Nesta fase, o componente React se prepara para sua inicialização, configurando os estados iniciais e props padrões se houverem.
- **Montagem:** Depois de preparar com todas as necessidades básicas, estado e props, o nosso Componente React está pronto para ser montado no DOM do navegador.
 - **componentWillMount()** É executado quando o componente estiver prestes a ser montado no DOM da página. Assim, após esse método ser executado o componente irá criar o nó no navegador. Todas as coisas que você deseja fazer antes do componente ser montado, devem ser definidas aqui.
 - **componentDidMount()** Este é o método que é executado depois que o componente foi montado no DOM

Ciclos de vida dos Componentes

- **Atualização:** Esta fase começa quando o componente React já nasceu no navegador e cresce recebendo novas atualizações. O componente pode ser atualizado de duas maneiras, através do envio de novas props ou a atualização do seu estado.
 - **componentDidUpdate()** é chamado imediatamente após a atualização.
 - **componentWillUpdate()** É executado somente quando `shouldComponentUpdate` devolver `true`.

Ciclos de vida dos Componentes

- **Desmontagem:** Nesta fase, o componente não é mais necessário e será desmontado do DOM. O método que se chama nesta fase é o:
 - `componentWillUnmount()`.

Exemplo

```
constructor(props){  
  super(props);  
  this.state={  
    hora:"00:00:00"  
  }  
}
```

Exemplo

```
componentDidMount(){
```

```
  setInterval(()=>{
```

```
    this.setState({hora: new Date().toLocaleTimeString()})
```

```
  },1000);
```

```
}
```


Exemplo

```
render(){  
  return (  
    <div>  
      <h1>Hora:</h1>  
      <h1>{this.state.hora}</h1>  
    </div>  
  );  
}
```

Eventos: Detalhando um pouco mais

- Vamos construir um componente chamado Membro e que tem uma propriedade chamada nome: `<Membro nome="seu nome">`
- Na sua aplicação você deve incluir dois botões com os rótulos : Logar como "Manoel" (coloque seu nome) e Sair.
- Ao acionar o evento onClick do botão Logar, a mensagem Bem-vindo Manoel deve ser exibida.
- Ao acionar o evento onClick do botão Sair, a mensagem deve ser apagada.

Eventos: Detalhando um pouco mais

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

```
class Membro extends Component{
```

```
  constructor(props){
```

```
    super(props);
```

```
    this.state = {
```

```
      nome:props.nome
```

```
    }
```

```
  }
```

Eventos: Detalhando um pouco mais

```
logar(){  
    let state = this.state;  
    state.nome="Manoel";  
    this.setState(state);  
}
```

Eventos: Detalhando um pouco mais

```
render(){  
  return(  
    <div>  
      <h2>Seja Bem-Vindo {this.state.nome}</h2>  
      <button onClick={this.logar.bind(this)}>Logar Como</button>  
      <button onClick={() => this.setState({nome: ""})}>Sair</button>  
    </div>  
  );  
}  
} export default Membro;
```

Eventos: Detalhando um pouco mais

- E se quisermos passar parâmetro para o método `logar`?

```
logar(valor){  
    let state = this.state;  
    state.nome=valor;  
    this.setState(state);  
}
```

Eventos: Detalhando um pouco mais

- A chamada muda para método com parâmetro. É preciso escrever uma função anônima.

```
render(){  
  return(<div>  
    <h2>Seja Bem-Vindo {this.state.nome}</h2>  
    <button onClick={() => this.logar("João")}>Logar Como</button>  
    <button onClick={() => this.setState({nome:""})}>Sair</button>  
  </div>  
  );  
}  
}export default Membro;
```

Renderização condicional

- Em muitas situações a renderização de um componente pode ser condicionada. Vamos ver um exemplo a seguir:

```
class App extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      status: false  
    }  
  }  
}
```


Renderização condicional

```
logar(status) {  
  this.setState({ status: status });  
}
```

Renderização condicional

```
render() {  
  return (  
    <div>  
      {this.state.status ?  
        <div>  
          <h1>Olá Manoel, </h1>  
          <button onClick={()=>this.logar(false)}>Logout</button>  
        </div>  
        :  
        <div>  
          <h1>Olá Visitante, Faça seu Login:</h1>  
          <button onClick={()=>this.logar(true)}>Login</button>  
        </div>  
      }  
    </div>  
  )  
}
```

Listas

- Vamos criar uma aplicação que exibe uma lista de objetos.
- A lista contém esses dados:
- `feed=[`

`{id:1, nome:'Manoel', curtidas:34, comentarios:23},`

`{id:2, nome:'Maria', curtidas:45, comentarios:12},`

`{id:3, nome:'Joao', curtidas:102, comentarios:70},`

`]`

Listas

1. Vamos criar um componente chamado Feed que tem as propriedades id, nome, comentário e curtidas:

```
function Feed(props){  
  return(  
    <div key={props.id}>  
      <h1>{props.nome}</h1>  
      <a>{props.curtidas} curtidas</a>  
      <a>{props.comentarios} comentarios</a>  
      <hr/>  
    </div>  
  );  
}  
export default Feed;
```

Listas

2. No App.js vamos usar o nosso Feed:

```
constructor(props){  
  super(props);  
  this.state={  
    feed:[  
      {id:1, nome:'Manoel', curtidas:34, comentarios:23},  
      {id:2, nome:'Maria', curtidas:45, comentarios:12},  
      {id:3, nome:'Joao', curtidas:102, comentarios:70},  
    ]  
  }  
}
```

Listas

```
render() {  
  return (  
    <div>  
      {this.state.feed.map((item) => {  
        return (  
          <Feed id={item.id} nome={item.nome}  
            curtidas={item.curtidas}  
            comentarios={item.comentarios} />  
        );  
      })}
```

Formulários

- A manipulação de formulários é algo muito comum nas aplicações web.
- Em React isso é uma atividade que envolve basicamente a manipulação de estados e de eventos.
- Vamos construir uma aplicação que permita a um usuário digitar seu peso e sua altura e com isso calcular o seu IMC. ($\text{IMC} = \text{Peso} / \text{Altura}^2$)

Formulários

```
class IMC extends Component{  
  constructor(props){  
    super(props);  
    this.state = {  
      peso:o,  
      altura:o,  
      imc:""  
    }  
  }  
}
```


Formulários

```
calcular(){  
    let state= this.state;  
  
    let imc=state.peso/state.altura**2;  
  
    state.imc=imc;  
  
    this.setState(state);  
}
```

Formulários

```
render(){
  return(
    <div>

      Peso: <input type="text" name="peso" value={this.state.peso} onChange={(e)=>
this.setState({peso:e.target.value})}/>

      Altura: <input type="text" name="altura" value={this.state.altura} onChange={(e)=>
this.setState({altura:e.target.value})}/>

      <button onClick={this.calcular.bind(this)}>Calcular</button>

      <br/>

      <h1>{this.state.imc}</h1>

    </div>

  );
}
```

Formulários

- Se o formulário tiver MUITOS campos, criar uma função diferente para tratar o evento onChange pode significar muito trabalho. Podemos "generalizar" essa função. Primeiro vamos agrupar em um objeto os campos do form:

```
this.state = {  
  form:{  
    peso:o,  
    altura:o,  
    imc:""  
  }  
}
```

Formulários

- Basta agora escrever uma função dadosForm generica e usar o nome do input:

```
dadosForm(e){  
  let state =this.state;  
  state.form[e.target.name]=e.target.value;  
  this.setState(state);  
}
```

Projeto: Biscoito da Sorte

- Vamos construir um projeto do zero chamado biscoito da sorte.
- A aplicação consiste em exibir uma imagem de um biscoito da sorte e um botão.
- Ao clicar em um botão, uma frase de boa sorte deve ser exibida na tela
- Baixe essa imagem
e coloque dentro de
uma pasta `/src/assets` (crie a pasta)



Projeto: Biscoito da Sorte

- Vamos usar o componente App
- Armazene as frases a seguir em um vetor no App
 - A vida trará coisas boas se tiveres paciência.
 - Demonstre amor e alegria em todas as oportunidades e verás que a paz nasce dentro de você.
 - Não compense na ira o que lhe falta na razão.
 - Defeitos e virtudes são apenas dois lados da mesma moeda.
 - A maior de todas as torres começa no solo.
 - Não há que ser forte.
- Crie um objeto (state) com a variável fraseSorteada:
`this.state = {fraseSorteada: 'Sortear uma frase'};`

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

import './App.css';

import biscoito from './assets/biscoito.png';

class App extends Component {

  constructor(props) {

    super(props);

    this.frases = ['A vida trará coisas boas se tiveres paciência.',

      'Demonstre amor e alegria em todas as oportunidades e verás que a paz nasce dentro de você.',

      'Não compense na ira o que lhe falta na razão.',

      'Defeitos e virtudes são apenas dois lados da mesma moeda.',

      'A maior de todas as torres começa no solo.',

      'Não há que ser forte.'

    ];

    this.state = {

      fraseSorteada:''

    };
  }
}
```

```
abrirBiscoito(){

    let numero = Math.floor(Math.random()*this.frases.length);

    this.setState({fraseSorteada:this.frases[numero]});

}

render() {

    return (

        <div className="container">

            <img alt="biscoito" src={biscoito} className="img"/>

            <button onClick={this.abrirBiscoito.bind(this)}>Sortear</button>

            <h1 className="textoFrase">{this.state.fraseSorteada}</h1>

        </div>

    );

}
```


Projeto: Biscoito da Sorte

- A estilização do projeto no App.css:

Mudar a cor de fundo do body para cinza:

```
body{  
    background-color: #DDD;  
}
```

Projeto: Biscoito da Sorte

- A estilização do projeto no App.css:

Disponer a imagem, o botão e a frase um abaixo do outro, centralizados e com largura de 100%:

```
.container{  
  display: flex;  
  flex-direction: column;  
  width: 100%;  
  align-items:center;  
  justify-items: center;  
}
```

Projeto: Biscoito da Sorte

- A estilização do projeto no App.css:

Largura e altura da imagem em 250px

```
.img{  
  width: 250px;  
  height: 250px;  
}
```

Projeto: Biscoito da Sorte

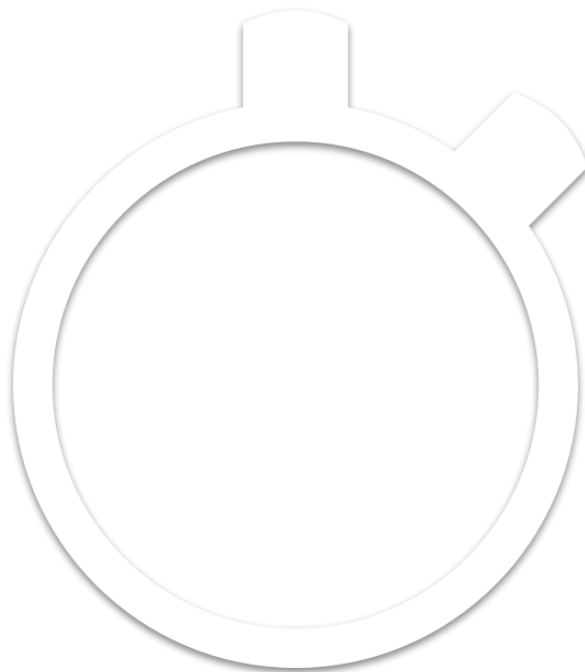
- A estilização do projeto no App.css:

Cor, estilo e alinhamento do texto

```
.textoFrase{  
    font-style: italic;  
    color: #dd7b22;  
    text-align: center;  
}
```

Projeto Cronômetro

- Vamos criar um cronômetro web. Primeiro coloque essa imagem numa pasta chamada /src/assets:



Projeto Cronômetro

```
<div className="container">  
  <img src={cronometro} className="img" />  
  <a className="timer">0.0</a>  
  <div className="areaTexto">  
    <button className="botao">Iniciar</button>  
    <button className="botao">Limpar</button>  
  </div>  
</div>
```

Projeto Cronômetro: Estilização

```
.container{  
  display: flex;  
  width: 100%;  
  align-items: center;  
  justify-content: center;  
  flex-direction: column;  
}
```

```
.img{  
  margin: 100px auto;  
  width: 300px;  
  height: 350px;  
}
```

Projeto Cronômetro: Estilização

```
.timer{  
    font-size: 130px;  
    margin-top: -330px;  
}
```

```
.areaTexto{  
    margin-top: 120px;  
    display: flex;  
}
```


Projeto Cronômetro: Estilização

```
.botao{  
    width: 150px;  
    background-color:thistle;  
    font-size: 20px;  
    text-align: center;  
    border-radius: 5px;  
    margin-left: 15px;  
    margin-right: 15px;  
    cursor: pointer;  
    padding: 5px;  
}
```

```
.botao:hover{  
    opacity: 0.9;  
    color:chartreuse;  
    transition: all 0.60s;  
}
```

Projeto Cronômetro: Procedural

```
import React, { Component } from 'react';  
  
import cronometro from './assets/cronometro.png';  
  
import './App.css';  
  
class App extends Component {  
  constructor(props) {  
    super(props);  
  
    this.state = {  
      tempo: 0.0,  
      botao: 'Vai'  
    };  
  
this.funcaoTimer = null;
```

Projeto Cronômetro: Procedural

```
vai() {  
  let estado = this.state;  
  if (this.funcaoTimer == null) {  
    estado.botao = 'Pausar';  
    this.funcaoTimer = setInterval(() => {  
      estado.tempo += 0.1;  
      this.setState(estado);  
    }, 100);  
  }  
}
```

```
else {  
  estado.botao = 'Vai';  
  clearInterval(this.funcaoTimer);  
  this.funcaoTimer = null;  
  this.setState(estado);  
}  
}
```

Projeto Cronômetro: Procedural

```
limpar() {  
  clearInterval(this.funcaoTimer);  
  this.setState({ tempo: 0 });  
}
```

Projeto Cronômetro: Procedural

```
<div className="container">
```

```
  <img src={cronometro} className="img" />
```

```
  <a className="timer">{this.state.tempo.toFixed(1)}</a>
```

```
  <div className="areaTexto">
```

```
    <button className="botao" onClick={this.vai.bind(this)}>{this.state.botao}</button>
```

```
    <button className="botao" onClick={this.limpar.bind(this)}>Limpar</button>
```

```
  </div>
```

```
</div>
```

API Hooks

- Até agora para manipular estado e ciclo de vida de um componente nós sempre usamos class based components
- Hooks é uma API que foi adicionada ao React 16.8. Bem nova considerando que o react está na versão 17.0.2 (em 30/05/2022)
- O que é um Hook?
 - Um Hook é uma função especial que te permite utilizar recursos do React.
 - Eles permitem, por exemplo, que você use o state e outros recursos do React sem escrever uma classe.
 - Reduz a verbosidade do código
- Pq só agora? Muitos projetos legados não usam Hooks (api é nova)...

A hook useState

- Vamos criar um App que adiciona Strings escritas num input na tela depois que o usuário clica em um botão.
- Vamos fazer isso primeiro usando um class based component e depois vamos refazer usando um function based com Hooks.
- Crie um componente chamado Tarefas

A hook useState

```
class Tarefa extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      texto: "",  
      lista: ['lavar os pratos', 'arrumar a cama']  
    }  
  }  
}
```


A hook useState

```
addTarefa() {  
    let estado = this.state;  
    estado.lista.push(this.state.texto);  
    estado.texto = "";  
    this.setState(estado);  
}
```

A hook useState

```
render() {  
  return (  
    <div><ul>{this.state.lista.map((item) =><li>{item}</li>)}</ul>  
    <input type="text" value={this.state.texto} onChange={(e) => this.setState({ texto: e.target.value })} />  
    <button onClick={this.addTarefa.bind(this)}>Incluir</button>  
  </div>  
  );  
}
```

A hook useState

- Vamos refazer o componente, dessa vez usando function based e hooks
- A **useState**, recebe um valor inicial para um estado e retorna um estado e uma função que modifica esse estado.
- Ex: `const [texto, setTexto] = useState('valor inicial do estado texto');`

A hook useState

```
import React, {useState} from "react";
```

```
function Tarefa2() {
```

```
  const [texto, setTexto] = useState('Digite sua tarefa');
```

```
  const [lista, setLista] = useState(['Lavar roupa', 'Varrer quarto']);
```

```
  function addTarefa(){
```

```
    setLista([texto,...lista]);
```

```
    setTexto("");
```

```
  }
```

A hook useState

```
return (  
  <div><ul>  
    {lista.map((item) =><li>{item}</li>)}  
  </ul>  
  <input type="text" value={texto} onChange={(e) => setTexto(e.target.value)} />  
  <button onClick={addTarefa.bind(this)}>Incluir</button>  
  </div>  
);
```

A hook useEffect

- A função `useEffect` tem potencial para substituir todas as funções de ciclo de vida de um componente (`componentDidMount`, `componentWillUpdate`, etc.).
- Ela recebe dois parâmetros: `useEffect` (função, lista)
 - lista: array com os estados a serem monitorados
 - função: uma função anônima que é executada a cada vez que algum dos estados monitorados muda.
- Vamos incluir o `useEffect` no nosso componente `Tarefa2`

A hook useEffect

```
import React, {useState, useEffect} from "react";

function Tarefa2() {

  const [texto, setTexto] = useState('Digite sua tarefa');

  const [lista, setLista] = useState(['Lavar roupa', 'Varrer quarto']);

  useEffect(()=>{

    localStorage.setItem('terfa',JSON.stringify(lista));

    },[lista]);
```

A hook useEffect

- Se o useEffect for chamado com a lista de estados vazia, o "efeito" será igual ao componentDidMount.

```
useEffect(()=>{  
    alert('componete acabou de ser montado');  
}, [ ] );
```


A hook useMemo

- Essa hook permite que uma função seja executada apenas quando um estado monitorado é modificado.
- Isso é muito útil pois evita que, possivelmente cálculos ou rotinas complexas sejam executados na renderização de um componente (que podem acontecer várias vezes).
- A useMemo recebe como parâmetros função anônima e uma lista de estados monitorados.
- A useMemo devolve o valor (o retorno) da função anônima.

A hook useMemo

```
let tamanho = useMemo(()=>{  
    return lista.length;  
},[lista]);
```

no return..

<h1>Você tem {tamanho} tarefas!</h1>

A hook useCallback

- Semelhante ao useMemo.
- Retorna uma função e não o valor de retorno como no Memo
- Podemos substituir o nosso **addTarefa** por um callBack:

```
function addTarefa(){  
  setLista([texto,...lista]);  
  setTexto("");  
}
```

```
const addTarefa = useCallback(()=>{  
  setLista([texto,...lista]);  
  setTexto("");  
  },[texto,lista]);
```

Requisições HTTP e JSON

- Uma das rotinas que mais caracterizam uma aplicação web são as requisições HTTP (request/response HTTP).
- Elas são a base da comunicação entre o front-end (lado cliente) e back-end (lado servidor).
- Um uso comum em React para requisições HTTP incluem permitir que o front-end:
 - Envie dados para o back-end (que podem ser persistidos)
 - Receba e exiba dados do back-end.
- Os dados armazenados em um back-end seguem um formato específico: XML, JSON, etc.
- Aqui na nossa disciplina vamos usar JSON por ser um dos padrões mais usados na atualidade e ainda por ser um padrão JavaScript.

Requisições HTTP e JSON

- JSON (JavaScript Object Notation) é um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto JavaScript.
- É comumente usado para transmitir dados em aplicativos da Web (por exemplo, enviar alguns dados do servidor para o cliente, para que possam ser exibidos em uma página da Web ou vice-versa).
- Mesmo que se assemelhe à sintaxe literal do objeto JavaScript, ele pode ser usado independentemente do JavaScript, e muitos ambientes de programação possuem a capacidade de ler (analisar) e gerar JSON.

Requisições HTTP e JSON

- O JSON existe como uma string — útil quando você deseja transmitir dados por uma rede.
- Ele precisa ser convertido em um objeto JavaScript nativo quando você quiser acessar os dados.
- Isso não é um grande problema — o JavaScript fornece um objeto JSON global que possui métodos disponíveis para conversão entre os dois.
- Um objeto JSON pode ser armazenado em seu próprio arquivo, que é basicamente apenas um arquivo de texto com uma extensão de .json, e um MIME type de application/json.

Requisições HTTP e JSON

- Para acessar dados armazenados em um formato JSON de um servidor, basta acessar a sua url. Em outras palavras, realizar uma requisição HTTP para a url específica do servidor.
- Por exemplo: [clique aqui](#)

Requisições HTTP e JSON : Exemplo

```
[ { "id": 1, "titulo": "Refeições proteicas para fazer antes de dormir",  
  "capa": "https://sujeitoprogramador.com/nutriapp/wp-content/uploads/2017/12/Screenshot_3-2.jpg",  
  "categoria": "Deita",  
},  
{ "id": 2, "titulo": "Cafeína – Bom ou Ruim ?",  
  "capa": "https://sujeitoprogramador.com/nutriapp/wp-content/uploads/2017/12/Screenshot_2-3.jpg",  
  "categoria": "Dicas",  
}, ]
```


Requisições HTTP e JSON : Exemplo

- Vamos construir um componente que acessa a url do slide anterior e exiba os dados baixados.
- Crie um projeto chamado requisicoeshttp e nele um componente (funcao + hooks) chamado Nutricao.
- O componente deve carregar os dados assim que for montado para depois exibir.
- Use o useState e useEffect

Requisições HTTP e JSON : Exemplo

```
import { useEffect, useState } from "react";

function Nutricao(){
  const [dicas, setDicas] = useState([]);

  useEffect(()=>{
    let url="https://sujeitoprogramador.com/rn-api/?api=posts";
    fetch(url).then((response)=>response.json()).then((myjson)=>{
      setDicas(myjson);
    })
  },[]);
}
```

Requisições HTTP e JASON : Exemplo

```
<header>
  <strong>React Nutri</strong>
</header>
{dicas.map((item)=>{
  return(
    <article key={item.id} className="post">
      <strong className="titulo">{item.titulo}</strong>
      <img src={item.capa} alt={item.titulo} className="capa" />
      <p className="subtitulo">
        {item.subtitulo}
      </p>
      <a className="botao">Acessar</a>
    </article>
  )
})}
```

Requisições HTTP e JASON : fetch API

- A API Fetch fornece uma interface JavaScript para acessar e manipular partes do pipeline HTTP, tais como os pedidos e respostas.
- Ela também fornece o método global `fetch()` que fornece uma maneira fácil e lógica para buscar recursos de forma assíncrona através da rede.
- O método `fetch ()` tem um argumento obrigatório, o caminho para o recurso que deseja obter (url).
- Ele retorna uma promessa que resolve a `Response` para uma requisição, seja ele bem-sucedido ou não.
- Fetch também provê um lugar lógico único para definir outros conceitos relacionados ao protocolo HTTP como CORS e extensões ao HTTP

Requisições HTTP e JASON : fetch API

```
fetch('flowers.jpg').then(function(response) {

    if(response.ok) {

        response.blob().then(function(myBlob) {

            var objectURL = URL.createObjectURL(myBlob);

            myImage.src = objectURL;

        });

    } else {

        console.log('Network response was not ok.');
```

Requisições HTTP e JSON : axios API

- O Axios é um cliente HTTP leve baseado no serviço \$http dentro do Angular.js v1.x e é semelhante à API Fetch nativa do JavaScript
- É baseado em promessas, o que oferece a capacidade de aproveitar o async e await do JavaScript para um código assíncrono mais legível.
- Também é possível interceptar e cancelar solicitações, e existe uma proteção integrada do lado do cliente contra a falsificação de solicitações entre sites
- Para instalar o axios, na pasta do seu projeto : **npm install axios**

Requisições HTTP e JSON : axios API

```
import axios from 'axios';  
  
....  
axios.get(url).then(response=>{  
    let myjson = response.data;  
    setDicas(myjson);  
}  
).catch((erro)=>{  
    alert('Deu erro'+ erro);  
})
```

Requisições HTTP e JSON : axios API

```
import axios from 'axios';
```

```
....
```

```
const user = {
```

```
  name: this.state.name
```

```
};
```

```
axios.post(url, { user })
```

```
.then(res => {
```

```
  console.log(res);
```

```
  console.log(res.data);
```

```
})
```


Requisições HTTP e JSON : axios API

- Usando uma instância base no Axios
- Em um arquivo chamado API.js :

```
import axios from 'axios';
```

```
export default axios.create({
```

```
  baseURL: `http:// sua url base/`
```

```
});
```

Requisições HTTP e JSON : axios API

```
import axios from 'axios';  
  
import API from '../api';  
  
API.delete(`users/${this.state.id}`)  
  
  .then(res => {  
    console.log(res);  
    console.log(res.data);  
  })
```

Trabalhando com Rotas em React

- Como escrever uma aplicação em React que permita exibir uma página (Componente) a partir de uma URL específica. Ex:
<http://localhost:3000/sobre> ou <http://localhost:3000/home>
- A solução para isso é usar o react-router-dom
- Para instalar digite no terminal: **npm install react-router-dom**
- Para entender como funcionam as rotas, crie 3 componentes : **Home, Sobre e Erro.**
- Cada componente deve inicialmente exibir um h1 com uma frase qualquer. Agrupe os componentes em um diretório chamado de pages.
- O objetivo é que cada componente seja exibido a partir de uma URL específica.

Trabalhando com Rotas em React

- Crie um diretório chamado de Rotas e nele um index.js
- Importe os componentes do react-router-dom:

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
```

- Crie um componente chamado Rotas (function based).

```
function Rotas() {  
  return ( <AQUI Vamos manipular as rotas> );  
} export default Rotas;
```

Trabalhando com Rotas em React

- O **BrowserRouter** permite implantar uma ou mais rotas dentro de uma aplicação.
- O **Route** permite escolher uma componente específico baseado em uma url/path.
- O **Routes** garante que apenas um Route será carregado de cada vez.

```
<BrowserRouter>
```

```
  <Routes>
```

```
    <Route exact path="/" element={Home} />
```

```
    <Route path="/sobre" element={Sobre} />
```

```
    <Route path="*" element={Erro}/>
```

```
  </Routes>
```

```
</BrowserRouter>
```

Trabalhando com Rotas em React

- Agora é só importar o componente Rotas e usar no index.

```
import Rotas from './rotas.js';
```

```
ReactDOM.render(  
  <React.StrictMode>
```

```
    <Rotas />
```

```
  </React.StrictMode>,  
  document.getElementById('root')
```

```
);
```

Trabalhando com Rotas em React

- `<Route exact path="/" element={Home} />` aqui o "exact" garante que apenas exatamente a url "/" ex: "<http://algumsite/>" vai exibir o componente Home.
- `<Route path="*" element={Erro} />` o path="*" indica que para qualquer url diferente das que estão nas rotas será exibido o componente Erro.

Trabalhando com Rotas em React

- Componentes que são exibidos via **Route** podem usar o componente **Link**
- Por exemplo, podemos definir um componente Menu:

```
<div >
```

```
  <Link to="/">Home</Link>
```

```
  <Link to="/sobre">Sobre</Link>
```

```
</div>
```


Trabalhando com Rotas em React

- É possível passar parâmetro em um path
- Suponha que queremos passar um id na página sobre
- **É só colocar ":id" depois do path "/sobre". Assim: path="/sobre/:id"**

```
<BrowserRouter>
```

```
  <Routes>
```

```
    <Route exact path="/" element={Home} />
```

```
    <Route path="/sobre/:id" element={Sobre} />
```

```
    <Route path="*" element={Erro}/>
```

```
  </Routes>
```

```
</BrowserRouter>
```

Trabalhando com Rotas em React

```
import { useParams, Link } from "react-router-dom";

function Sobre(){

  const {id} = useParams();

  return(

    <div>

      <h1>Essa é a página Sobre com id = {id}</h1>

      <Link to="/"> Home </Link>

    </div>

  );

}
```

Práticas

- Depois de tudo que vimos até agora, chegou a hora de praticar um pouco mais.
- Vamos montar uma aplicação web para resolver as questões da lista encontrada [aqui](#)
- Além de responder a lista, você deve seguir um padrão:
 - Criar um Menu onde cada item corresponde a uma das questões da lista.
 - Criar um componente para cada questão (Questao1, Questao2,...).
 - Incluir (além da solução) o enunciado da questão em cada componente.

Práticas: Filmaria

- Vamos montar uma aplicação web para exibir detalhes de uma lista de Filmes obtidos em servidor REST
- A lista de filmes encontrada [aqui](#)
- Além disso os detalhes de um filme pode ser obtido usando o ID depois da url. Por exemplo [assim](#).
- Use o mesmo modelo do site de Dicas de Nutrição.

Práticas: Filmaria

- Crie um Componente Header que deve ser exibido sempre (Como se fosse um Menu horizontal)
- Crie uma página Home para exibir a lista (Título, Capa e um botão Acessar).
- Use rotas do react router Dom
- Use o axios para conexão HTTP.

Práticas: Filmaria

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

```
import './styles.css';
```

```
export default function Header(){
```

```
  return(
```

```
    <header>
```

```
      <Link className="logo" to="/">Filmaria</Link>
```

```
      <Link className="favoritos" to="/">Salvos</Link>
```

```
    </header>
```

Práticas: Filmaria

```
header{  
  display: flex;  
  align-items: center;  
  justify-content: space-around;  
  width: 100%;  
  height: 60px;  
  background-color: brown;  
}
```

```
.logo{  
  text-decoration: none;  
  font-size: 30px;  
  color: #FFF;  
  cursor: pointer;  
}
```

Práticas: Filmaria

```
.favoritos{  
    text-decoration: none;  
    font-size: 30px;  
    color: #000;  
    border-radius: 5px;  
    font-weight: bold;  
    padding: 5px;  
    cursor:pointer;  
    background-color: #FFF;  
}
```


Práticas: Filmaria

```
import axios from 'axios';  
  
////https://sujeitoprogramador.com/r-api/?api=filmes  
  
//Base URL: https://sujeitoprogramador.com/  
  
//Rota TODOS OS FILMES: r-api/?api=filmes  
  
// Filme com ID: r-api/?api=filmes/556  
  
const api = axios.create({  
  baseURL:'https://sujeitoprogramador.com'  
});  
  
export default api;
```

Práticas: Filmaria

```
import api from '../services/api.js';

import {useEffect, useState} from 'react'

import './index.css'

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

function Home() {

  const [filmes, setFilmes]=useState([]);

  useEffect(()=>{

    async function loadFilmes(){

      const response = await api.get('/r-api/?api=filmes');

      setFilmes(response.data);

    }

    loadFilmes();

  },[]);
```

```
    return (

      <div className="container">

        <div className="lista-filmes">

          {filmes.map((item)=>{

            return (

              <article key={item.id}>

                <strong>{item.nome}</strong>

                <img src={item.foto} alt={item.nome}/>

                <Link to="/">Acessar</Link>

              </article>

            );

          })}

        </div>

      </div>

    );

  }
```

Práticas: Filmaria

```
.container{  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    justify-content: center;  
    width: 800px;  
}  
.lista-filmes{  
    margin: 20px;  
}
```

```
.lista-filmes article{  
    display: flex;  
    flex-direction: column;  
    justify-content: center;  
    align-items: center;  
    margin: 15px;  
    background-color: #fff;  
    border-radius: 5px;  
    padding: 15px;  
}
```

Práticas: Filmaria

```
.lista-filmes strong{
    padding-bottom: 15px;
    font-size: 22px;
}

.lista-filmes img{
    width: 900px;
    max-height: 350px;
}
```

```
.lista-filmes article a{
    text-decoration: none;
    color: #FFF;
    background-color: brown;
    font-size: 25px;
    width: 100%;
    text-align: center;
    padding-top: 10px;
    padding-bottom: 10px;
    border-bottom-left-radius: 5px;
    border-bottom-right-radius: 5px;
}
```

Práticas: Filmaria

- Agora vamos detalhar cada Filme criando um componente page de mesmo nome.
- Primeiro passo é criar uma rota para cada filme com seu ID específico:

```
<Route exact path="/filme/:id" element={<Filme/>}/>
```

- Na página Home é preciso alterar o Link de "/" para `{`/filme/${item.id}`}`. Assim cada filme em Home terá um "Acessar" com o id correto de seu respectivo filme.
- Componente toast : [Detalhes aqui](#)
 - `import {toast} from 'react-toastify'`

Práticas: Filmaria

- Como recuperar o id passado no Link?
 - A função useParams do react-router-dom

```
export default function Filme() {  
  
  const { id } = useParams();  
  
  const [filme, setFilme] = useState({});  
  
  const [loading, setLoading] = useState(true);  
  
  const history = useNavigate();
```

Práticas: Filmaria

```
useEffect(() => {  
  async function loadFilme() {  
    const response = await api.get('/r-api/?api=filmes/' + id);  
    if(response.data.length===0){  
      history('/'); // manda para Home  
      return;  
    }  
    setFilme(response.data);  
    setLoading(false);  
  }  
  loadFilme();  
}, [history,id]);
```

Práticas: Filmaria

```
function salvarFilme(){  
    let filmesSalvos = JSON.parse(localStorage.getItem("filmes")) || [];  
    let hasFilme = filmesSalvos.some((item) => item.id === filme.id);  
    if(hasFilme){  
        toast.error('Você já Salvou esse filme');  
        return;  
    }  
    filmesSalvos.push(filme);  
    localStorage.setItem("filmes", JSON.stringify(filmesSalvos));  
    toast.success('Filme salvo com sucesso');  
}
```


Práticas: Filmaria

```
if (loading) {  
    return (  
        <div className="filme-info">  
            <h1>carregando filme... </h1>  
  
        </div>  
    );  
}
```

```
else{  
    return (  
        <div className="filme-info">  
            <h1> {filme.nome} </h1>  
  
            <img src={filme.foto} alt={filme.nome}/>  
  
            <h3>Sinopse</h3>  
  
            {filme.sinopse}  
  
            <div>  
  
                <button onClick={()=>salvarFilme()}>Salvar</button>  
  
                <button>  
  
                    <a target="blank" href={`https://www.youtube.com/results?search_query=${filme.nome} trailer`}>Trailer</a>  
  
                </button>  
  
            </div>  
  
        </div>  
    );  
}
```

Práticas: Filmaria

```
.filme-info{  
    display: flex;  
    flex-direction: column;  
    max-width: 900px;  
}
```

```
.filme-info img{  
    border-radius: 20px;  
}
```

```
.filme-info button{  
    margin-right: 5px;  
    margin-top: 15px;  
    cursor: pointer;  
    border-radius: 5px;  
    border: 0;  
    font-size: 20px;  
    outline: none;  
    padding: 12px;  
}
```

Práticas: Filmaria

```
.filme-info button:hover{  
    background-color: brown;  
    color: #FFF;  
}
```

```
.filme-info a{  
    text-decoration: none;  
    color: #000;  
}
```

```
.filme-info a:hover{  
    background-color: brown;  
    color: #FFF;  
}
```

Práticas: Filmaria

- Próximo Passo é Criar o componente de Filmes Salvos: Favoritos

```
export default function Favoritos() {
```

```
  const [salvos, setSalvos] = useState([]);
```

```
  useEffect(() => {
```

```
    let filmes = JSON.parse(localStorage.getItem("filmes")) || [];
```

```
    setSalvos(filmes);
```

```
  }, []);
```

Práticas: Filmaria

```
function excluirFilmes(id) {  
    let filmes = salvos.filter((item) => {  
        return item.id !== id;  
    });  
    setSalvos(filmes);  
    localStorage.setItem("filmes", JSON.stringify(filmes));  
    toast.success('Filme Excluído com Sucesso!')  
}
```

Práticas: Filmaria

```
return (  
  <div  
    className="meus-filmes">  
      <h1>Meus Filmes</h1>  
  
      {salvos.length === 0  
        ? <span> Você não possui  
        filmes salvos :(</span>)  
      }  
  
      <ul>
```

```
{salvos.map(item) => {  
  
  return (  
  
    <li key={item.id}>  
  
      <span> {item.nome} </span>  
  
      <div>  
  
        <Link to={`/filme/${item.id}`}>Ver Detalhes</Link>  
  
        <button onClick={() => excluirFilmes(item.id)}>Excluir</button>  
  
      </div>  
  
    </li>  
  
  );  
  
  })  
}</ul>  
  
</div>  
  
);
```

Práticas: Filmaria

```
.meus-filmes{  
    display: flex;  
    flex-direction: column;  
    width: 100%;  
    align-items: center;  
    justify-content: center;  
}
```

```
ul{  
    padding: 0;  
}
```

```
ul li{  
    padding: 0;  
    list-style: none;  
    min-width: 600px;  
    display: flex;  
    align-items: center;  
    justify-content: space-between;  
}
```

```
span{  
    font-size: 23px;  
}
```

Práticas: Filmaria

```
.button{  
    margin-left: 10px;  
}
```

```
ul li div a{  
    text-decoration: none;  
    color: brown;  
}
```


Firestore

- O que é o Firestore e que problemas ele ajuda a resolver?
- É um serviço do google e não é completamente gratuito. Vamos usar apenas o que for free :)
- O React serve para construir o Front-end o Firestore ajuda nas questões de Back-end:
 - Banco de Dados
 - autenticação
 - segurança
 - outros

Firestore

- Como acessar? <http://firebase.google.com>
- Fazer login com sua conta google
- Clique em " ir para o console "
- Serão listados os seus projetos Firestore além da uma opção "add projeto"
- Clique em "add projeto"
- escolha um nome...aqui eu escolhi INF012.
- Depois ative o analytics (se não estiver ativado) e continue
- Escolha o país (Brasil) e criar projeto
- Você será levado ao painel do seu projeto

Firestore

- O painel do seu projeto tem um menu lateral com todas as funções que podem ser usadas em sua aplicação como autenticação, cadastro de usuários, armazenamento em nuvem, entre outros

Autenticação com Firebase

- Uma excelente funcionalidade do firebase é a autenticação
- Você pode cadastrar usuários ou usar usuários cadastrados de serviços como google, github, entre outros.
- Vamos ver como cadastrar usuários.
- No menu lateral do Firebase, no item autenticação, aba "sign-in method" habilite a opção de email e senha.

Firestore

- Crie um projeto básico e mantenha apenas App/index.js.
- Instale no seu app a biblioteca do firestore: **npm install firestore**
- Nas configurações do seu projeto (app) no Firestore, copie as credenciais de acesso.
- Essas ao lado são as MINHAS
- No App.js inclua dois inputs (email e senha) e um botão cadastrar e os respectivos states.

```
var firestoreConfig = {  
  
  apiKey: "AIzaSyApr4ms-z9mNxBg6R-LrcgefKfvWqminFT",  
  
  authDomain: "info12.firebaseio.com",  
  
  projectId: "info12",  
  
  storageBucket: "info12.appspot.com",  
  
  messagingSenderId: "541302410071",  
  
  appId: "1:541302410071:web:7bd66803460ce0e9c60c12",  
  
  measurementId: "G-80oSXYDMSY"  
  
};
```

Autenticação com Firebase

- Crie dentro de src um arquivo chamado firebaseConnection.js

```
import { initializeApp } from
"firebase/app"

import { getAuth } from "firebase/auth";

const firebaseConfig = {

  //cole aqui as suas credenciais

};

var auth = null;

const app = initializeApp(firebaseConfig);
```

```
if(app){
  auth = getAuth();
}

export default auth;
```

Autenticação com Firebase

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

```
const [senha, setSenha]=useState("");
```

```
.....
```

```
<label>Email: </label>
```

```
  <input type="text" value={email} onChange={(e)=>setEmail(e.target.value)}/>
```

```
<label>Senha: </label>
```

```
  <input type="password" value={senha} onChange={(e)=>setSenha(e.target.value)}/>
```

```
<button onClick={()=>{novoUsuario()}}>Cadastrar</button>
```

Autenticação com Firebase

- Próximo passo é implementar a função novoUsuario()
- Import isso dentro do App.js

```
import auth from "../fireBaseConnection";
```

```
import {createUserWithEmailAndPassword} from "firebase/auth";
```


Autenticação com Firebase

```
async function novoUsuario(){  
  await  
  createUserWithEmailAndPassword( auth ,email,senha)  
  .then((value)=>{  
    console.log(value);  
  })  
}
```

```
.catch((error)=>{  
  if(error.code==='auth/weak-password'){  
    alert('Senha Fraca');  
  }else  
  if(error.code==='auth/email-already-in-use'){  
    alert('Email já em uso');  
  }  
})  
}
```

Autenticação com Firebase

- A lista de erros do firebase auth pode ser vista aqui:
- <https://firebase.google.com/docs/reference/js/firebase.auth.Auth>
- Agora que já cadastramos usuários no Firebase vamos entender como monitorar o login usando usuários cadastrados.
- Quando cadastramos um usuário no firebase ele faz o login automaticamente.
- Vamos então começar fazendo logout. Inclua um botão e uma função chamada logout:
- `<button onClick={()=>{logout()}}>Logout</button>`

```
async function logout(){  
  await signOut(auth).then(()=>{  
    alert("usuario desconectado");  
  });  
}
```

Autenticação com Firebase

- Podemos monitorar um login via **useEffect** e **onAuthStateChanged** para saber se algum usuário está logado no seu navegador
- Primeiro crie dois states:
- `const [user, setUser]=useState(false);`
- `const [loggedUser, setLoggedUser]=useState({});`

Autenticação com Firebase

```
useEffect(()=>{  
  async function checkLogin(){  
    onAuthStateChanged(auth, (user)=>{  
      if(user){  
        setUser(true);  
        setLoggedUser({  
          uid:user.uid,  
          email:user.email  
        })  
      }  
    })  
  }  
})
```

```
}else{  
  setUser(false);  
  setLoggedUser({});  
}  
})  
}  
checkLogin();  
},[]);
```

Autenticação com Firebase

```
<h1>FIREBASE + REACT</h1><br/>
```

```
{user {{ (
```

```
  <div>
```

```
    <strong>Seja Bem-Vindo!</strong>
```

```
    <span>{loggedUser.uid} - {loggedUser.email}</span>
```

```
  </div>
```

```
)}
```

Autenticação com Firebase

- Vamos por fim fazer um login
- Inclua um botão chamado fazer login
- `<button
onClick={()=>{login()}}>Fazer
Login</button>`

```
async function login(){  
  await firebase.auth()  
    .signInWithEmailAndPassword(email,senha)  
    .then((value)=>{  
      console.log(value);  
    })  
    .catch((error)=>{  
      console.log(error);  
      alert(error.message);  
    })  
}
```

Autenticação com Firebase (gmail)

- Vamos por fim fazer um login
- Inclua um botão chamado fazer login
- `<button
onClick={()=>{login()}}>Fazer
Login</button>`

```
async function login(){  
  var provider = new GoogleAuthProvider();  
  provider.addScope('profile');  
  provider.addScope('email');  
  signInWithPopup(auth, provider).then(function  
(result) {  
    // This gives you a Google Access Token.  
    var name = result.user.displayName;  
    // The signed-in user info.  
    var user1 = result.user;  
    console.log(name);  
    console.log(user1);  
  });  
}
```

Context API

- Como definir um state "global"?
- Serve por exemplo para "guardar as credenciais de acesso de um usuário e exibir em vários componentes de um uma Aplicação"
- A Context API do react permite definir contextos globais que garantem o acesso a states em diferentes componentes.
- Vamos definir um contexto para usuários como exemplo.
- Crie uma pasta contexts e nela um arquivo chamado user.js

Context API

```
import {useState, createContext} from 'react'

export const UserContext = createContext({});

function UserProvider({children}){
  const [user, setUser]=useState({});
  return(
    <UserContext.Provider value={{user,setUser}}>
      {children}
    </UserContext.Provider>
  );
}

export default UserProvider;
```

Context API: Espalhando o contexto para todos os componentes

```
import Rotas from './routes';  
  
import UserProvider from './contexts/user';  
  
function App() {  
  return (  
    <UserProvider>  
      <div>  
        <Rotas/>  
      </div>  
    </UserProvider>  
  );  
}  
export default App;
```

Context API: Usando o contexto em outro componente

```
import Rotas from './routes';

import {UserContext} from '../contexts/user'

function Login() {

  // No lugar de const [user, setUser]=useState({});

  const {user, setUser}=useContext(UserContext);

  return (

    <div>


      {user}

    </div>

  );

}export default Login;
```

Sistema de Chamados: Login




Entrar

Acessar

[Criar uma conta](#)

Sistema de Chamados: Novo Usuário



The image shows a user registration form for a ticket system. At the top, there is a dark blue header with a light blue icon of a person holding a key. Below the header, the title "Nova Conta" is centered. The form consists of four input fields: "Seu nome", "email@email.com", and a password field with five asterisks. A dark blue "Cadastrar" button is positioned below the fields. At the bottom, there is a link that says "Já possui uma conta? Entre aqui!".

Nova Conta


Seu nome


email@email.com


Cadastrar


Já possui uma conta? Entre aqui!


Sistema de Chamados: Perfil




 Chamados

 Clientes

 Configurações

 Meu perfil



Nome

Manoel Carvalho Marques Neto



Email


manoelnetom@gmail.com


Salvar


Sair


Sistema de Chamados: Clientes



 Chamados

 Clientes

 Configurações

 Clientes

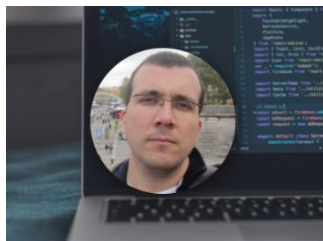

Nome


CNPJ


Endereço


Salvar


Sistema de Chamados: Chamados






 Chamados

 Clientes

 Configurações

 **Atendimentos**



CLIENTE	ASSUNTO	STATUS	CADASTRADO EM	#
Sujeito	Suporte	Em aberto	20/06/2021	 

Sistema de Chamados : Passo a Passo

Veja o tutorial aqui!