

Desenvolvimento de Software para WEB

Aula 01
Introdução ao REACT

Preparando o ambiente

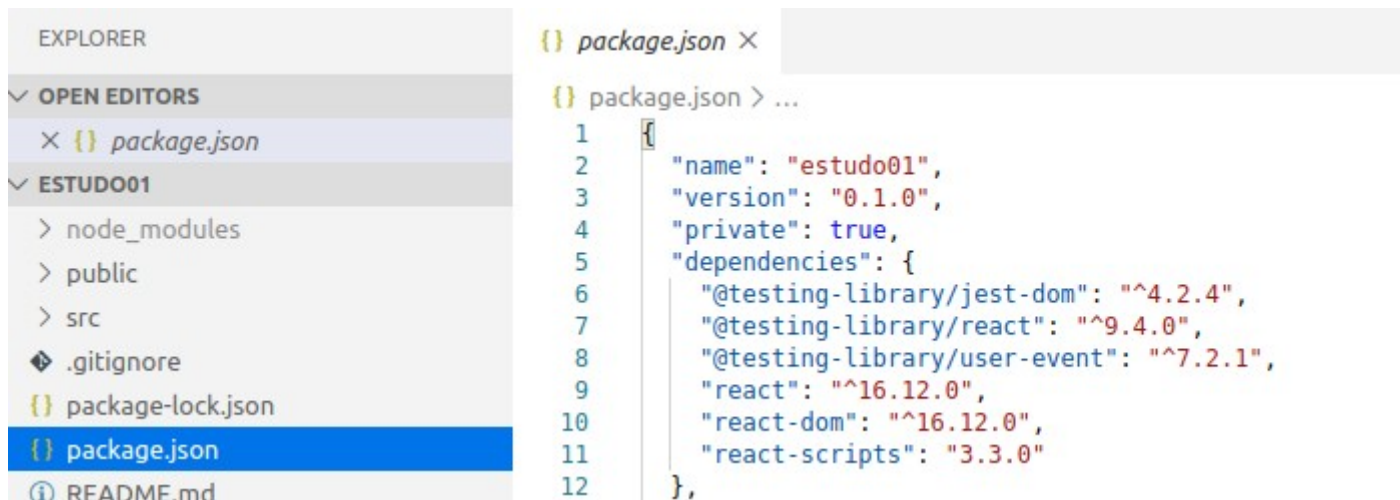
- instale o Node.js <https://nodejs.org/en/>
- instale o MongoDB <https://www.mongodb.com/>
- instale o VSCode <https://code.visualstudio.com/>
- Cheque suas versões:
 - node -v
 - npm -v

Primeiro projeto

- Abra o terminal para criar o projeto, e digite:
 - **sudo npm i -g create-react-app**
- **O create-react-app** é um programa que auxilia na criação de aplicações React, configurando o projeto de forma descomplicada. Outra alternativa é utilizar o **npx**.
- Agora devemos criar nosso projeto:
 - **create-react-app aula01**
- Agora entre na pasta e inicie o projeto:
 - **cd aula01**
 - **npm start**

Primeiro projeto

- package.json

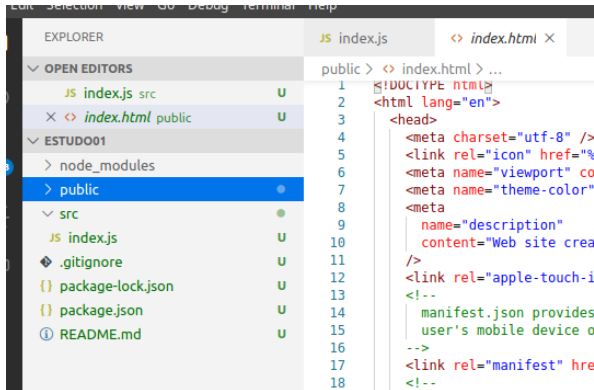


```
EXPLORER
├── OPEN EDITORS
│   └── {} package.json
├── ESTUDO01
│   ├── > node_modules
│   ├── > public
│   ├── > src
│   ├── .gitignore
│   ├── {} package-lock.json
│   └── {} package.json
└── README.md

{} package.json > ...
1  {
2    "name": "estudo01",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^4.2.4",
7      "@testing-library/react": "^9.4.0",
8      "@testing-library/user-event": "^7.2.1",
9      "react": "^16.12.0",
10     "react-dom": "^16.12.0",
11     "react-scripts": "3.3.0"
12   },
```

Primeiro projeto

- Em qualquer browser, seu projeto estará no endereço: <http://localhost:3000>
- Agora, abra a pasta do projeto com o VSCode, e apague TODO o conteúdo da pasta src.



Primeiro projeto

- Em src, crie o arquivo **index.js** e digite o código:

```
import ReactDOM from 'react-dom';
```

```
const root = document.getElementById('root'); //código vindo do index.html  
ReactDOM.render("Olá mundo!",root);
```

Primeiro projeto

- index.html (criado pelo projeto) - SPA

```
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
```

Primeiro projeto

- Agora, tente executar o código:

```
import ReactDOM from 'react-dom';
```

```
const root = document.getElementById('root'); //código vindo do index.html  
ReactDOM.render(<h1>Olá mundo!</h1>,root);
```

- Ocorreu algum erro?
- Devemos importar a biblioteca React.

Primeiro projeto

- Usando o **React**, para fazer funcionar código **JSX**:

```
import React from 'react';  
import ReactDOM from 'react-dom';  
  
const root = document.getElementById('root');  
//ReactDOM.render("Olá mundo!",root);  
ReactDOM.render(<h1>Jefferson</h1>,root);
```

Primeiro projeto

- O código JSX facilita escrever código semelhante ao HTML. Caso contrário, você deverá incluir **manualmente** o código HTML!
- Veja como ficaria, sem JSX:

```
const root = document.getElementById('root');
```

```
let h2 = document.createElement('h2');
```

```
let texto = document.createTextNode('Olá mundo!');
```

```
h2.appendChild(texto);
```

```
root.appendChild(h2);
```

Primeiro projeto

- Exercício:
- Crie, sem usar JSX e dentro de index.js, o código que renderiza a seguinte página HTML:
 - 1-) Superman
 - 2-) Batman
 - 3-) Wonder Woman
- Depois, crie o mesmo código agora usando JSX.
- Dica, use as tags e .
- Caso seu computador ainda não esteja configurado, use:
 - <https://repl.it/languages/reactjs>

Componentes - Conceito

- O uso de componentes.
 - Um componente é um “pedaço” de software independente, reusável o qual pode ser usado para construir aplicações maiores.
 - Em React, podemos criar componentes em forma de **funções** ou **classes**.

Componentes – Olá Mundo!

- No projeto anterior, crie uma pasta dentro de src chamada “**components**”. Usaremos essa pasta para organizar nosso código em componentes. Obviamente, ela não é obrigatória.
- Dentro de components, crie o arquivo **OlaMundo.jsx**
- A extensão **.jsx** não é obrigatória, mas é uma convenção.

Componentes – Olá Mundo!

- **OlaMundo.jsx**

```
import React from 'react';

function olaMundo() {
  return <div>
    <h1>Olá Mundo!</h1>
  </div>
}

export default olaMundo;
```

Componentes – Olá Mundo!

- Arquivo index.js

```
import React from 'react';  
import ReactDOM from 'react-dom';  
  
import OlaMundo from './components/OlaMundo';  
  
const root = document.getElementById('root');  
ReactDOM.render(<OlaMundo/>,root);
```

Componentes – Olá Mundo! (Arrow)

- Criando um componente sem nome usando uma função arrow (mais usado):

```
export default ()=>{  
  return <h1>Olá Mundo!</h1>  
}
```

- OU:

```
export default ()=>  
  <div>  
    <h1>Olá Mundo!</h1>  
  </div>
```


Componentes - Props

- Propriedades (ou o famoso **props**).
 - Torna possível passar “parâmetros” para os nossos componentes. Por exemplo, em OlaMundo.jsx:

```
import React from 'react';

export default (props) =>
  <div>
    <h1>Olá {props.nome}!</h1> //tudo que está entre { } é interpretado como
                                //um código javascript!
  </div>
```

Componentes - Props

- Já em index.js, caso eu queira passar um valor para a propriedade “nome”:

```
import React from 'react';  
import ReactDOM from 'react-dom';
```

```
import OlaMundo from './components/OlaMundo';  
const root = document.getElementById('root');
```

```
ReactDOM.render(<OlaMundo nome='Jefferson'/>,root);
```

Componentes - Props

- Exercício:
- Crie um componente chamado “Estudante.jsx”. Passe, via **props**, o nome, idade, curso e cidade natal do estudante. Mostre os dados na tela usando o JSX `` e ``.

Componentes – Múltiplos

- Múltiplos componentes em um único arquivo.
- Podemos criar um arquivo JSX formado por vários componentes.
- Atenção! Não usaremos o **export default**!
- Crie o arquivo Vingadores.jsx, dentro da pasta **components**.

Componentes – Múltiplos

- Vingadores.jsx

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

```
const CapitaoAmerica = props => //nova sintaxe  
  <h1> Olá {props.nome}, eu sou o Capitão América. </h1>
```

```
const ViuvaNegra = props =>  
  <h1> Olá {props.nome}, eu sou a Viúva Negra. </h1>
```

```
export {CapitaoAmerica,ViuvaNegra};
```

Atenção! Os nomes exportados devem ser usados exatamente como foram exportados!

Componentes – Múltiplos

- index.js

```
import React from 'react';
import ReactDOM from 'react-dom';

import { CapitaoAmerica, ViuvaNegra } from './components/Vingadores';

const root = document.getElementById('root');
ReactDOM.render(<div>
  <CapitaoAmerica nome='Jefferson' />
  <ViuvaNegra nome='Thomas' />
</div>, root);
```

Componentes – Múltiplos

- index.js (segunda forma, usando *aliases*)

```
import React from 'react';
import ReactDOM from 'react-dom';

import { CapitaoAmerica as Ca, ViuvaNegra as Vn } from './components/Vingadores';

const root = document.getElementById('root');
ReactDOM.render(<div>
  <Ca nome='Jefferson' />
  <Vn nome='Thomas' />
</div>, root);
```

Atenção! Os nomes aliases não podem começar com letras minúsculas.

Componentes – Erro Comum!

- Retornando múltiplos elementos dentro de um componente, ou seja, múltiplas tags JSX (elementos adjacentes).
- Seja o seguinte componente Herois.jsx:

```
import React from 'react';  
export default Herois = props =>  
  <h1>Capitão América</h1>  
  <h1>Homem-Aranha</h1>  
  <h1>Incrível Hulk</h1>
```

- Ele apresenta um **erro**, pois não devemos retornar elementos adjacentes sem um “pai”.

Componentes – Erro Comum!

- Existem 3 formas de **solucionar** o problema dos elementos múltiplos adjacentes:
 - usar o `<div>` (mais comum)
 - usar o `<> </>`
 - usar uma lista `[]`

Componentes – Erro Comum!

export default props =>

```
<div>  
  <h1>Capitão América</h1>  
  <h1>Homem-Aranha</h1>  
  <h1>Incrível Hulk</h1>  
</div>
```

export default props =>

```
<>  
  <h1>Capitão América</h1>  
  <h1>Homem-Aranha</h1>  
  <h1>Incrível Hulk</h1>  
</>
```

export default props =>[

```
  <h1>Capitão América</h1>,  
  <h1>Homem-Aranha</h1>,  
  <h1>Incrível Hulk</h1>  
]
```

Atenção! No caso do uso de listas, os elementos devem ser separados por vírgula!

Componentes – Relação Hardcoded

- Componentes em aplicação React se relacionam em uma estrutura de “árvore”, onde componentes podem ser formados por outros componentes (usualmente chamados de “filhos”)
- Crie dois componentes (arquivos .jsx):
 - Personagem.jsx
 - Casa.jsx

Componentes – Relação Hardcoded

- Personagem.jsx

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

```
export default props =>
```

```
  <div>
```

```
    {props.nome} da casa {props.casa}.
```

```
  </div>
```

Componentes – Relação Hardcoded

- Casa.jsx

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

```
import Personagem from './Personagem';
```

```
export default props =>
```

```
  <div>
```

```
    <Personagem nome='Arya' casa='Stark'/>
```

```
    <Personagem nome='Robert' casa='Baratheon'/>
```

```
    <Personagem nome='John Snow' casa='Targaryan'/>
```

```
  </div>
```

Agora basta chamar a tag <Casa> dentro do index.js

Componentes – Relação Genérica

- O código anterior é chamado de hardcoded pois ele “trava” o componente Casa em um determinado número de personagens. Caso o desenvolvedor queira modificar esse número, ele terá que obrigatoriamente modificar o código fonte de Casa.
- Uma solução elegante seria poder passar todos os personagens que eu quisesse, dentro da tag <Casa>. Isso é possível graças ao **props.children**
- Vamos modificar o arquivo Casa.jsx para:

Componentes – Relação Genérica

- Casa.jsx

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

```
export default props =>
```

```
  <div>
```

```
    {props.children}
```

```
  </div>
```

- Tudo que for passado dentro de <Casa> ? </Casa>, será renderizado no lugar de **props.children**.

Componentes – Relação Genérica

- Já em index.js

```
import React from 'react';
import ReactDOM from 'react-dom';

import Casa from './components/Casa';
import Personagem from './components/Personagem';

const root = document.getElementById('root');
ReactDOM.render(
  <div>
    <Casa>
      <Personagem nome="Arya" casa="Stark"/>
      <Personagem nome="Sansa" casa="Stark"/>
    </Casa>
  </div>, root)
```


Componentes – Relação Genérica

- Exercício
- Crie mais de uma `<Casa>` em `index.js`, com novos personagens para aquela Casa.
- Passe como `props.casa` para `<Casa casa=?>` o nome da Casa e coloque esse nome dentro de `<h1>`, no componente Casa, logo acima de `{props.children}`

Componentes – Propagando Props

- Agora a ideia é que os componentes filhos “herdem” as propriedades (props) do componente pai.
- No nosso exemplo, não queremos declarar a propriedade “casa” na tag <Personagem>. Iremos declará-la na tag <Casa> a qual irá repassar pros filhos.

Componentes – Propagando Props

- O componente Personagem.jsx:

```
export default props =>  
  <div>  
    {props.nome} da casa {props.casa}.  
  </div>
```

- Em index.js:

```
ReactDOM.render(  
  <div>  
    <Casa casa='Stark'>  
      <Personagem nome='Arya' />  
      <Personagem nome='Sansa' />  
    </Casa>  
  </div>, root);
```

Componentes – Propagando Props

- Em Casa.jsx

```
export default props =>
  <div>
    <h1>Casa {props.casa}</h1>

    {React.Children.map(props.children, personagem => {
      return React.cloneElement(personagem, { ...props });
    })}

  </div>
```

Componentes – Funções Internas

- Um componente, até o momento, é apenas uma única função que retorna uma conteúdo JSX.
- É possível, no entanto, criar um componente com mais funções internas.
- Vamos criar um componente IMC (índice massa corpórea) que irá receber via props o peso (em kg) e a altura (em metros).
- Dentro do componente, teremos uma função que calcula o imc e também iremos retornar o JSX mostrando o IMC.

Componentes – Funções Internas

- IMC.jsx (primeira versão)

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

```
export default props =>{
```

```
  return <div>
```

```
    <h3>Sua altura é {props.altura}m e seu peso é {props.peso}kg.</h3>
```

```
    <h3> seu IMC é ...</h3>
```

```
  </div>
```

```
}
```

Componentes – Funções Internas

- index.js (note a passagem de números)

```
import React from 'react';  
import ReactDOM from 'react-dom';  
  
import IMC from './components/IMC';  
  
const root = document.getElementById('root');  
ReactDOM.render(  
  <div>  
    <IMC altura={1.84} peso={90}/>  
  </div>, root);
```

Componentes – Funções Internas

- IMC.jsx (segunda versão)

```
export default props =>{  
  
  function calcularIMC(peso,altura){  
    return peso/(altura*altura);  
  }  
  
  return <div>  
    <h3>Sua altura é {props.altura}m e seu peso é {props.peso}kg.</h3>  
    <h3> seu IMC é {calcularIMC(props.peso, props.altura)}</h3>  
  </div>  
  
}
```


Componentes – Funções Internas

- IMC.jsx (terceira versão)

...

```
const calcularIMCV2 = props =>{  
  return props.peso/(props.altura*props.altura);  
}
```

```
return <div>  
  <h3>Sua altura é {props.altura}m e seu peso é {props.peso}kg.</h3>  
  <h3>seu IMC é {calcularIMC(props.peso, props.altura)}</h3>  
  <h3>seu IMC é {calcularIMCV2(props)}</h3>  
</div>
```

...

Componentes – Funções Internas

- Exercício
- Modifique o componente IMC.jsx anterior criando uma nova função que recebe como parâmetro o valor do IMC e retorna um JSX de acordo com a tabela abaixo:

Resultado	Situação
Abaixo de 17	Muito abaixo do <i>peso</i>
Entre 17 e 18,49	Abaixo do <i>peso</i>
Entre 18,50 e 24,99	<i>Peso</i> normal
Entre 25 e 29,99	Acima do <i>peso</i>
Entre 30 e 34,99	<i>Obesidade</i> I
Entre 35 e 39,99	<i>Obesidade</i> II (severa)
Acima de 40	<i>Obesidade</i> III (mórbida)

Componentes - Filho ► Pai

- A ideia aqui é que o componente filho (inserido dentro do arquivo do componente pai) passe uma mensagem para o componente pai.
- A solução é uma forma indireta: o pai passa para o filho uma função de notificação (assim como uma inscrição). Quando o filho quiser dizer algo ao pai, ele usa essa função passada como parâmetro.
- Crie dois componentes:
 - Filho.jsx
 - Pai.jsx

Componentes - Filho ► Pai

- Filho.jsx

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

```
export default props =>
```

```
  <div>
```

```
    <button onClick={() => props.notificarPai('...Tudo bom?')}>
```

```
      Oi Pai!
```

```
    </button>
```

```
  </div>
```

Componentes - Filho ► Pai

- Pai.jsx

```
import React from 'react';
import Filho from './Filho';

export default props => {
  const msgRecebida = msg => alert(`Recebi: ${msg}`);
  return (
    <div>
      <Filho notificarPai={msgRecebida} />
    </div>
  )
}
```

Agora basta chamar a tag <Pai> dentro do index.js

Componentes - Classes

- MinhaClass.jsx

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

export default class MinhaClasse extends Component{
  render(){
    return (
      <div>Oi, eu sou o {this.props.nome}</div>
    )
  }
}
```

Componentes - State

- Imagine que você queira criar um sistema de pontuação para determinadas cidades, onde o usuário clica em um botão e atribui um ponto a uma determinada cidade.
- Vamos fazer uma aplicação, o qual implementa essa lógica para apenas duas cidades.

Componentes - State

- CidadeSimples.jsx (versão função)

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

```
export default props =>{  
  let fortaleza=0,quixada=0
```

```
  return (  
    <div>
```

```
      <h3>
```

```
        Fortaleza: {fortaleza} </h3>
```

```
        <h3>Quixadá: {quixada}</h3>
```

```
        <button onClick={()=>fortaleza++}>Fortaleza</button>
```

```
        <button onClick={()=>quixada++}>Quixadá</button>
```

```
      </div>
```

```
    )
```

```
  }
```


Componentes - State

- CidadeSimples.jsx (versão classe 1)

```
export default class CidadeSimples extends Component{  
  render(){  
    let fortaleza = 0, quixada = 0  
    return (  
      <div>  
        <h3>Fortaleza: {fortaleza} </h3>  
        <h3>Quixadá: {quixada}</h3>  
        <button onClick={()=>fortaleza++}>Fortaleza</button>  
        <button onClick={()=>quixada++}>Quixadá</button>  
      </div>  
    )  
  }  
}
```

Componentes - State

- CidadeSimples.jsx (versão classe 2)

```
export default class CidadeSimples extends Component{

  constructor(){
    super()
    this.fortaleza = 0
    this.quixada = 0
  }

  render(){
    //let fortaleza = 0, quixada = 0
    return (
      <div>
        <h3>Fortaleza: {this.fortaleza} </h3>
        <h3>Quixadá: {this.quixada}</h3>
        <button onClick={()=>this.fortaleza++}>Fortaleza</button>
        <button onClick={()=>this.quixada++}>Quixadá</button>
      </div>
    )
  }
}
```

Componentes - State

- No exemplo anterior, perceba que não importa clicar no botão, que o valor das variáveis “fortaleza” e “quixadá” não irá mudar.
- A solução é criar estados (states) para variáveis que mudem de acordo com a aplicação.

Componentes - State

- Caso seu componente precise de variáveis que mudem de acordo com certas ações e que essas mudanças **definam o estado do componente**, você deverá usar o ***state***.
- Por exemplo, vamos novamente criar um componente, onde os usuários possa atribuir pontos a algumas cidades (desta vez, quatro cidades).
- Crie o componente Cidades.jsx

Componentes - State

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

export default class Cidades extends Component {

  state = { fortaleza: 0, quixada: 0, jericoacoara: 0, limoeiro: 0 }

  votaFor = () =>{
    this.setState({fortaleza:this.state.fortaleza+1});
  }

  /*
  mesma ideia pros outros...
  */

  render() {
    return (
      <div>
        <button onClick={this.votaFor}>Fortaleza: {this.state.fortaleza}</button>
        <button onClick={this.votaQui}>Quixadá: {this.state.quixada}</button>
        <button onClick={this.votaJeri}>Jericoacoara: {this.state.jericoacoara}</button>
        <button onClick={this.votaLimo}>Limoeiro do Norte: {this.state.limoeiro}</button>
      </div>
    );
  }
}
```

Componentes - State

- Inicializando o state (pode ser no construtor):

```
state = { fortaleza: 0, quixada: 0, jericoacoara: 0, limoeiro: 0 }
```

- Mudando o estado (setState):

```
votaFor = () =>{  
  this.setState({fortaleza:this.state.fortaleza+1});  
}
```

- Chamando a função no onClick do botão

```
<button onClick={this.votaFor}>Fortaleza: {this.state.fortaleza}</button>
```

Componentes - State

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

```
export default class CidadeSimples extends Component{
```

```
  constructor(){  
    super()  
    this.state = {fortaleza:0,quixada:0}  
  }
```

```
  render(){  
    return (  
      <div>  
        <h3>Fortaleza: {this.state.fortaleza} </h3>  
        <h3>Quixadá: {this.state.quixada}</h3>  
        <button onClick={()=>this.setState({fortaleza:this.state.fortaleza+1})}>Fortaleza</button>  
        <button onClick={()=>this.setState({quixada:this.state.quixada+1})}>Quixadá</button>  
      </div>  
    )  
  }  
}
```

Nessa versão estamos usando:

- construtor, inicializando o state

*- implementando o setState dentro do botão (recomendável)
caso seja algo simples.*

React Hooks

- Adição à versão 16.8 do React.
- <https://pt-br.reactjs.org/docs/hooks-overview.html>
- É possível que componentes baseados em funções agora tenha estado (state).
- Vejamos o exemplo de um contador simples:

React Hooks

- Contador.jsx

```
import React from 'react';

export default props =>{
  let cont = 0;
  return(
    <div>
      <h1>Contador: {cont}</h1>
      <button onClick={()=>++cont}> Incrementa
    </button>
    </div>
  )
}
```

Clicar no botão não irá alterar o estado interno do componente...

React Hooks

- Contador.jsx (com Hooks)

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

```
export default props =>{  
  const [cont,setCont] = useState(0);  
  return(  
    <div>  
      <h1>Contador: {cont}</h1>  
      <button onClick={()=>setCont(cont+1)}> Incrementa </button>  
    </div>  
  )  
}
```

<https://pt-br.reactjs.org/docs/hooks-overview.html>

React Hooks

- `useEffect`
- Função chamada SEMPRE que o componente é atualizado.
- Pode ser útil no caso do estado do componente mudar, uma ação ser tomada.
- Por exemplo, toda vez que o número contador mudar, iremos dizer se ele é par ou ímpar.

React Hooks

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

export default props =>{
  const [cont,setCont] = useState(0);
  const [status,parImpar] = useState('par');

  useEffect(()=>{
    if(cont%2===0)
      parImpar('par');
    else
      parImpar('impar')
  },[cont])

  return(
    <div>
      <h1>Contador: {cont}</h1>
      <h1>Status: {status}</h1>
      <button onClick={()=>setCont(cont+1)}> Incrementa </button>
    </div>
  )
}
```

React Hooks

- Exercício
- Reimplemente a votação de cidades, destacando a cidade mais votada e menos votada com o `useEffect`.