

# React JS

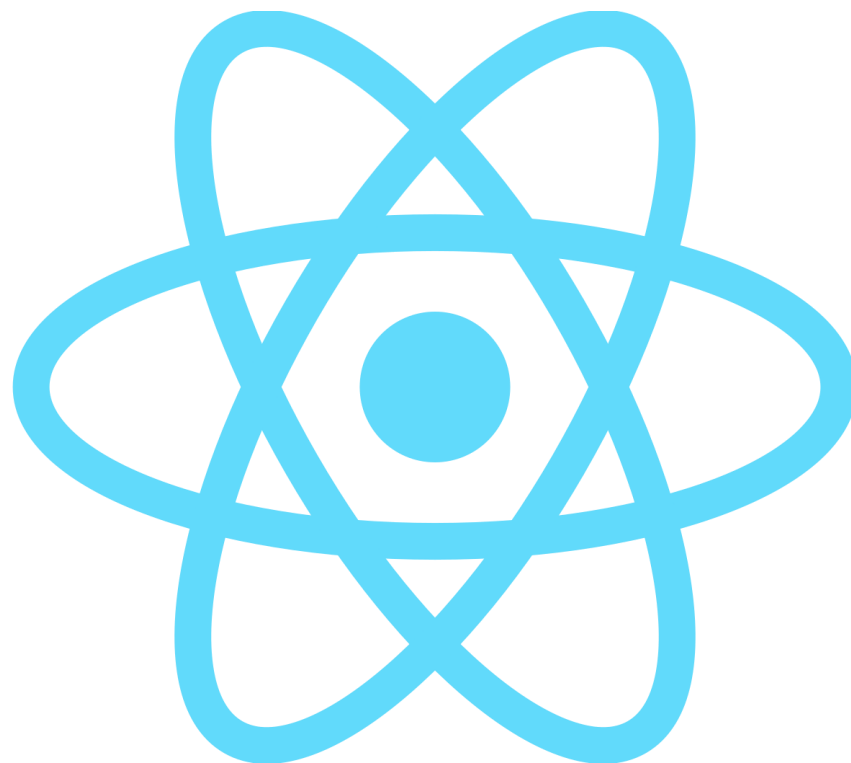
## Aula 04 e 05

Instrutor: Rodolfo Moura

# Intro

Revisão sobre  
componentes

Realização da  
atividade proposta 1



# PROPS

O que são?

Valores passados para os componentes

Dinamicidade dos componentes

Mudança do valor para execução

Passado como atributo para componente

Necessário resgatar dentro de uma propriedade/argumento chamada props na função de definição do componente.

Props são somente de leitura

# PROPS

Criação de componente e função

Adição de props dentro do parênteses da função

Adicionamos a propriedade {props.propriedade} dentro do código.

Exportamos a componente normalmente

```
function Item2(props) {
```

```
<div>  
  <h1>Componente teste {props.nome}</h1>  
</div>
```

# PROPS

“Chamamos” a propriedade dentro da tag que chamamos a componente (no arquivo app.js), colocando em aspas o valor desejado.

Replicando o componente e modificando sua propriedade teremos algo diferente cada vez

```
<Item2 nome="Extra" />
```

Edit src/App.js and save to reload.

**Componente teste Extra**

```
<Item2 nome="Extra" />  
<Item2 nome="Repetido" />  
<Item2 nome="De novo" />
```

Edit src/App.js and save to reload.

**Componente teste Extra**  
**Componente teste Repetido**  
**Componente teste De novo**

# PROPS

Podemos colocar também dados dinâmicos, e chamá-los da maneira como normalmente chamamos variáveis no React.

```
const nome = "Rodolfo"
```

```
<Item2 nome={nome} />
```

Também podemos trabalhar com componentes que possuem mais de uma propriedade.

# PROPS

Podemos colocar também dados dinâmicos, e chamá-los da maneira como normalmente chamamos variáveis no React.

```
const nome = "Rodolfo"
```

```
<Item2 nome={nome} />
```

Também podemos trabalhar com componentes que possuem mais de uma propriedade.

```
<Pessoa nome="Rodolfo" idade="30" />
```

# PROPS

Em caso de um objeto com muitas propriedades, podemos também chamar suas propriedades diretamente no () da função, sem mencionar “props” posteriormente no corpo do código.

Tratamos como um objeto, inclusive no uso de {}

```
function Pessoa ({nome, foto, idade, funcao})  
  return (
```



# PROPS

Além de atributos, nas nossas props nós também poderemos adicionar métodos, da mesma maneira como trabalhamos com objetos anteriormente.

Exemplo, temos a função no nosso código (em App.js):

```
function logando () {  
  | console.log("Teste")  
}
```

# PROPS

Fazemos algo no nosso componente que realize a chamada de uma função, nesse caso, em um botão:

```
<button size="small" onClick={props.funcao}>Botão</button>
```

Em seguida chamamos a função da mesma maneira que chamamos nossas props, ao realizar a chamada do componente:

```
funcao={logando}/>
```

# Validação de PROPS

Acontece quando fazemos o asseguração de que estaremos recebendo os tipos corretos de props nos nossos componentes. Evitando assim bugs e erros desnecessários.

Podemos fazer tao validação com o uso de propTypes.

# Validação de PROPS

## Usando PropTypes:

```
import React from 'react';
import propTypes from 'prop-types';
export default function Nomes(props) {
  return (
    <h1>Nome: {props.name}</h1>
  );
}

Nomes.propTypes = {
  name: propTypes.string.isRequired,
};
```

Dessa maneira aparecerá um erro no console caso a prop não seja string.

# PROPS

É comum trabalharmos com listas no react.

Pq seria mais produtivo trabalhar com elas para componentes?

Testar repetição de “cards”.

# PROPS

Uso de uma função do Javascript

```
{array.map()}
```

Uso de uma “transformação” =>

# PROPS

Uso de uma função do Javascript

```
{array.map()}
```

Uso de uma “transformação” =>

```
{array.map((dados) => (<Item titulo={dados}/>))}
```

# PROPS

Para múltiplos props, temos o exemplo:

```
var noticias = [  
  {titulo: "Titulo 1", sub: "Subtitulo 1"},  
  {titulo: "Titulo 2", sub: "Subtitulo 2"},  
  {titulo: "Titulo 3", sub: "Subtitulo 3"},  
  {titulo: "Titulo 4", sub: "Subtitulo 4"}];  
  
function App() {  
  return (  
    <div>  
      {noticias.map(  
        (dato) => (<Componente01 titulo={dato.titulo} sub={dato.sub}/>)}  
      )}  
    </div>  
  );  
}
```



# State

O estado da aplicação

Lugar onde os dados vem e se transformam com o tempo

2 categorias de componentes:  
Presentational e Container

Ou Stateless e Stateful

# State

## Presentational (Stateless)

Apenas apresentação de dados, daí não possuem estado. Ex.:

```
import React from 'react';

export default function Presentational() {
  return (
    <h1>Hello World</h1>
  );}
```

Normalmente a maioria de componentes será dessa categoria

# State

## Container (Stateful)

Não só possuem exposição de dados, mas também alguma lógica.

```
import React from 'react';

export default class Container extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'World' };
  }
  render() {
    return (
      <h1>Hello {this.state.name}</h1>
    );}}}
```

Não podem ser escritos como função, devem obrigatoriamente ser uma classe.

# State

## Container (Stateful)

Podemos ter situações também em que é possível alterar o estado do componente.

```
export default class Container extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { name: 'World' };  
  }  
  change(event) {  
    this.setState({ name: event.target.value });  
  }  
  render() {  
    return (  
      <div> <input value={this.state.name} onChange={(e) => this.change(e)} />  
      <p>Hello {this.state.name}!</p> </div>  
    );  
  }  
}
```

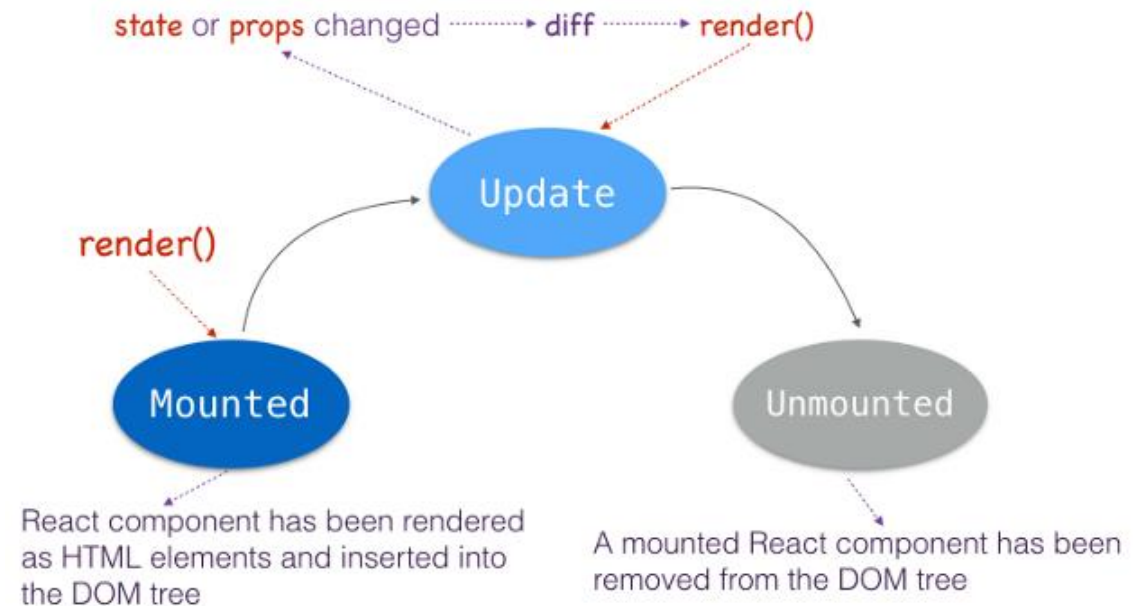
A cada mudança que ocorrer no input, um evento é disparado na função `change`. A função `change` por sua vez altera o estado do componente, usando a função `setState`, que é nativa do React.

Toda vez que o estado for alterado, o React automaticamente invoca de novo a função `render`, que irá renderizar a UI com os novos dados inputados pelo usuário.

# Lifecycle/ Ciclo de vida de Componente

Para componentes mais complexos, existem métodos que foram adicionados na API dos componentes do React.

Existem 3 Fases: Mounting, Updating e Unmounting (criação, atualização e destruição, respectivamente).



# Lifecycle/ Ciclo de vida de Componente

Temos os seguintes métodos do lifecycle dos componentes:

**componentWillMount**, executado logo antes do primeiro render. Não é muito usado, geralmente faz mais sentido simplesmente usar o próprio construtor da classe.

**componentDidMount**, executado logo após o primeiro render. É provavelmente o método mais usado. Alguns exemplos de casos de uso são: chamadas AJAX, manipulação do DOM, início de timeouts e setIntervals, etc.

**componentWillReceiveProps**, executado quando as props que o componente recebe são atualizadas. Geralmente é usado quando os componentes precisam reagir aos eventos externos.

**componentWillUpdate**, é igual ao **componentWillMount**, só que ele executa logo antes da atualização de um componente.

**componentDidUpdate**, é igual ao **componentDidMount**, só que ele executa logo depois da atualização de um componente.

**componentWillUnmount**, executado quando o ciclo de vida de um componente termina e ele vai ser removido do DOM. É muito usado para remover timeouts e setIntervals que foram adicionados.

**shouldComponentUpdate**, deve retornar true/false. Esse valor vai dizer que se o componente deve ser atualizado ou não, com base em certos parâmetros. Geralmente é usado para resolver questões de performance.

# Lifecycle/ Ciclo de vida de Componente

## Exemplo direto (Timer):

```
import React from 'react';

export default class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { elapsed: 0 };
    this.tick = this.tick.bind(this);
  }
  componentDidMount() {
    this.timer = setInterval(this.tick, 1000);
  }
  componentWillUnmount() {
    clearInterval(this.timer);
  }
  tick() {
    this.setState({ elapsed: this.state.elapsed + 1 });
  }
  render() {
    return <h1>{this.state.elapsed} seconds</h1>; } }
```

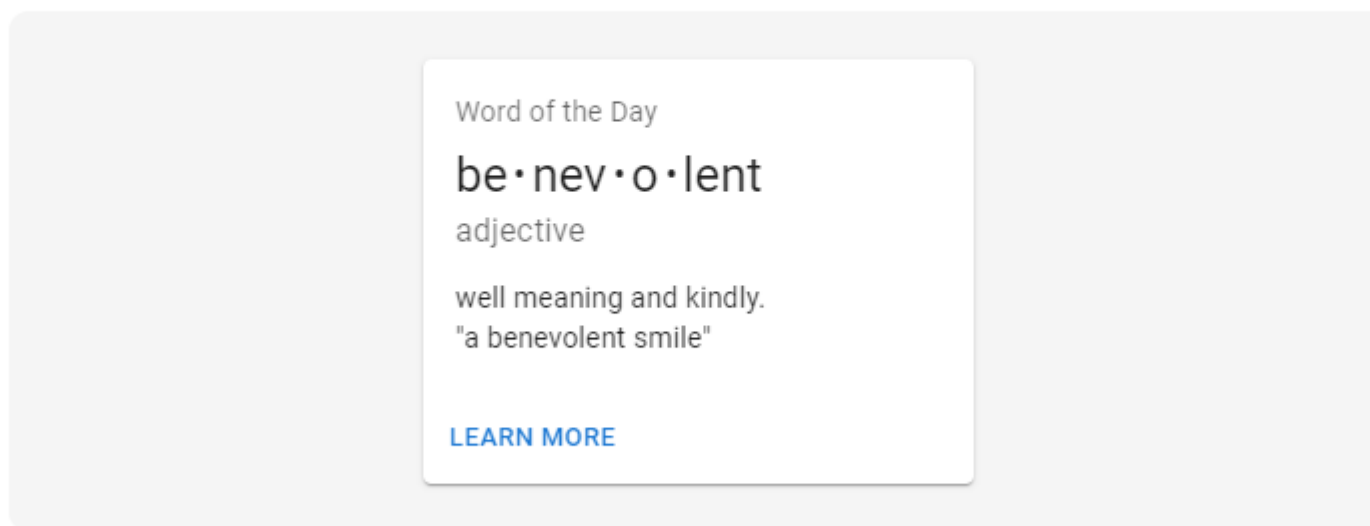


# Importação de Componentes e Bibliotecas

Podemos importar componentes prontos diretamente de sites, copiando e colando seus códigos.

## Basic card

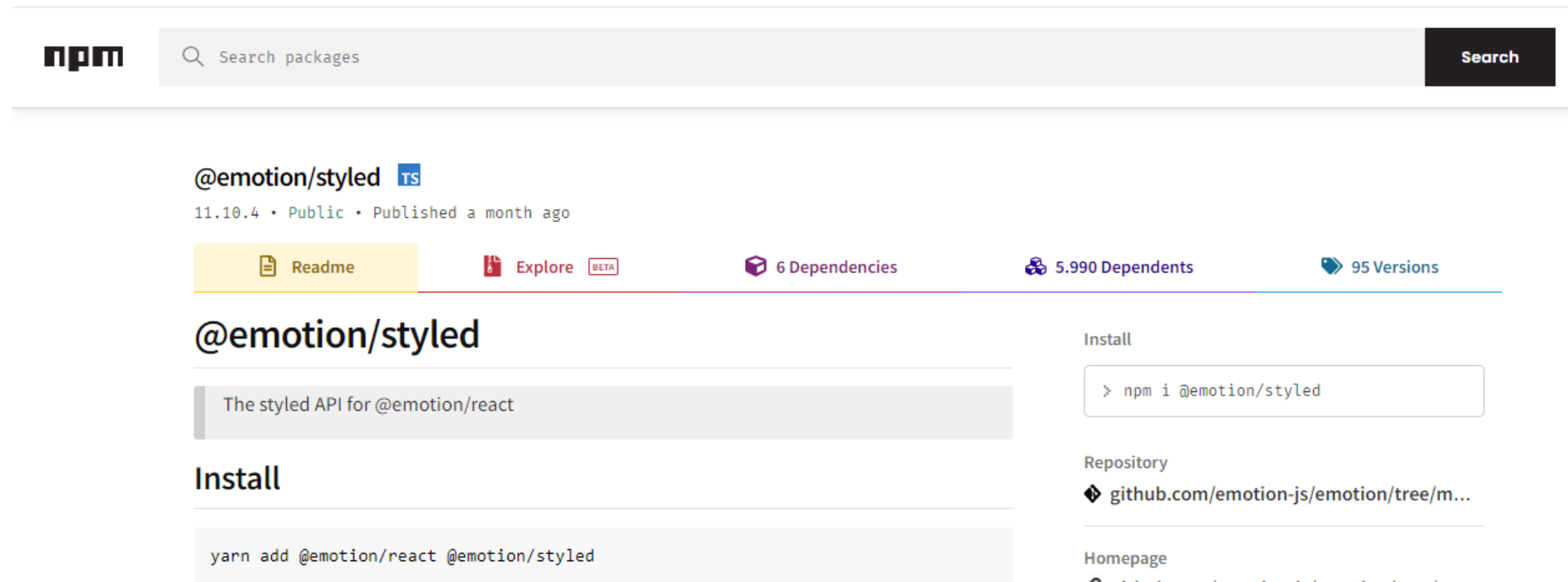
Although cards can support multiple actions, UI controls, and an overflow menu, use restraint and remember that cards are entry points to more complex and detailed information.





# Importação de Componentes e Bibliotecas

No entanto, é importante notar que alguns desses componentes requerem a instalação de bibliotecas, que podem ser achadas no site do npm, e instaladas com o comando `/npm i @"bibliotecaescolhida"`



The screenshot shows the npm package page for `@emotion/styled`. At the top, there's a search bar with the npm logo and a search button. Below the search bar, the package name `@emotion/styled` is displayed with a TypeScript (TS) icon. The version `11.10.4` is shown, along with the status `Public` and the publication date `Published a month ago`. A horizontal bar contains links for `Readme`, `Explore` (with a BETA tag), `6 Dependencies`, `5.990 Dependents`, and `95 Versions`. The main section features the package name `@emotion/styled` and a description: `The styled API for @emotion/react`. Under the `Install` heading, a code block shows the command `yarn add @emotion/react @emotion/styled`. On the right side, there's an `Install` section with a code block showing `> npm i @emotion/styled`, a `Repository` section with the link `github.com/emotion-js/emotion/tree/m...`, and a `Homepage` section with the same link.

npm

Search packages

Search

`@emotion/styled` TS

11.10.4 • Public • Published a month ago

Readme Explore BETA 6 Dependencies 5.990 Dependents 95 Versions

`@emotion/styled`

The styled API for `@emotion/react`

Install

`yarn add @emotion/react @emotion/styled`

Install

`> npm i @emotion/styled`

Repository

`github.com/emotion-js/emotion/tree/m...`

Homepage

`github.com/emotion-js/emotion/tree/m...`

# Atividade

- Buscar um componente (cartão) no site
- `https://mui.com/material-ui/react-card/`**
- Instalar as bibliotecas necessárias para uso da componente
- Criar um Array com 5 (CINCO) objetos que possuam as props que serão usadas na componente.
- Colocar um título na página e usar um background colorido.

**ENTREGA ATÉ 18H**