

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}/>)}
```

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>  
    );  
  }  
}
```

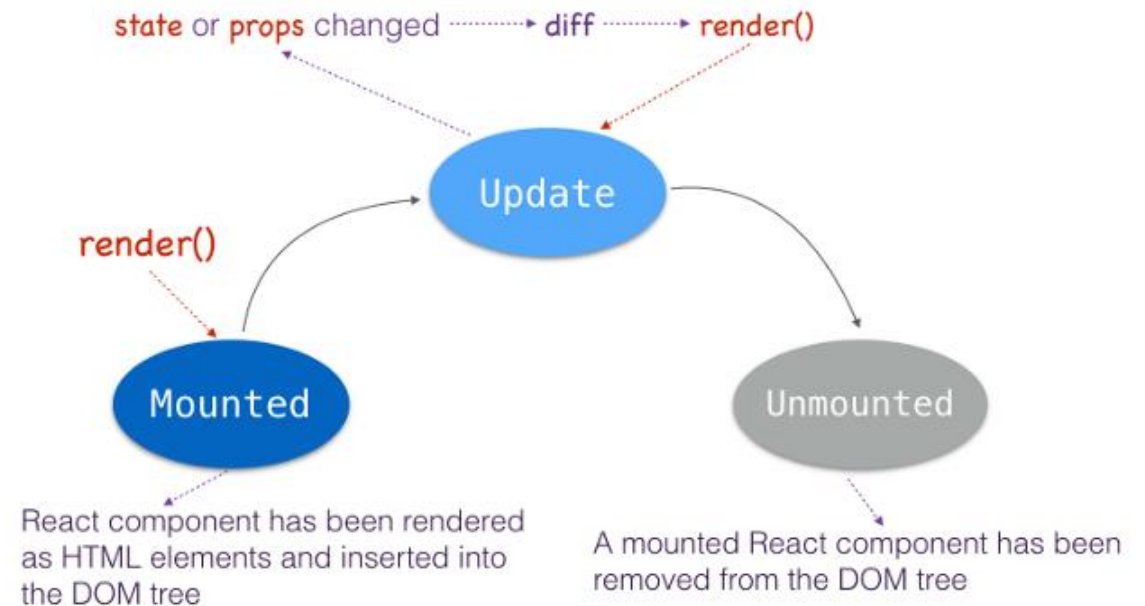
A cada mudança que ocorrer no input, um evento é disparado na função `change`. A função `change` por sua vez altera o estado do componente, usando a função `setState`, que é nativa do React.

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';

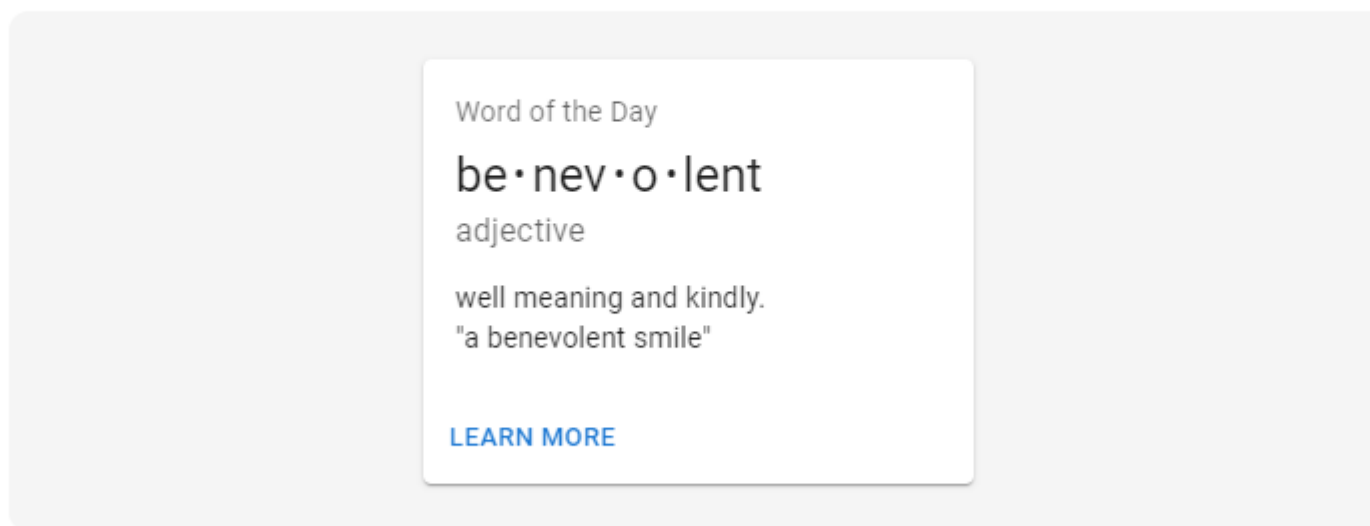
export default class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { elapsed: 0 };
    this.tick = this.tick.bind(this);
  }
  componentDidMount() {
    this.timer = setInterval(this.tick, 1000);
  }
  componentWillUnmount() {
    clearInterval(this.timer);
  }
  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.

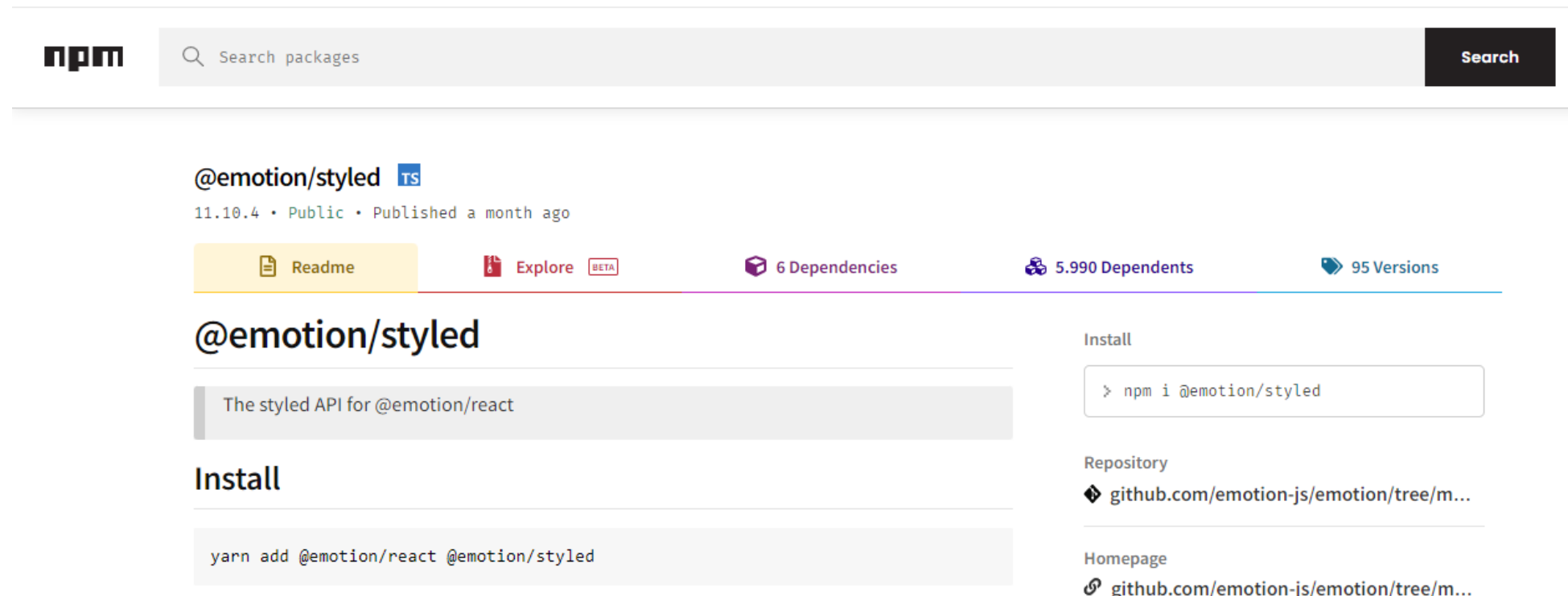
Basic card

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 as public and published a month ago. A horizontal bar contains links for Readme, Explore (marked BETA), 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 a link to `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...](https://github.com/emotion-js/emotion/tree/main)

Homepage

[github.com/emotion-js/emotion/tree/m...](https://github.com/emotion-js/emotion/tree/main)