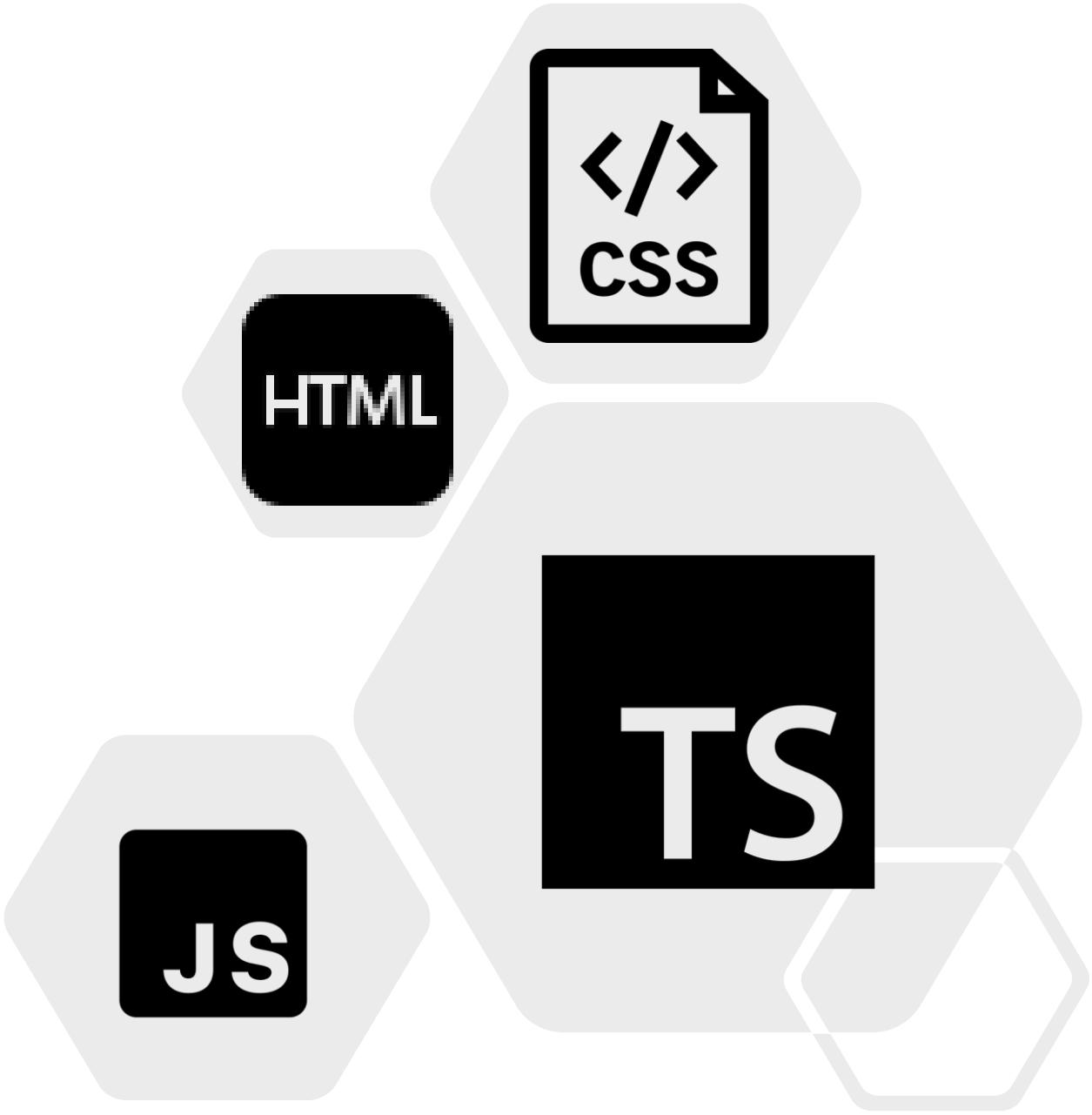


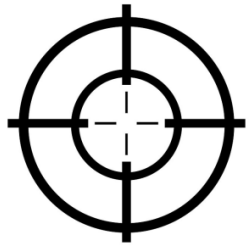
Componentes, propriedades, estado e ciclo de vida.

Componentes

Um componente é uma função ou classe com objetivo de renderizar informações para compor a interface disponível ao cliente.

Com componentes pode-se dividir a interface do cliente em partes independentes e reutilizáveis. Estas partes encaixam-se para montar a interface completa. Os componentes também são responsáveis por tratar as interações do cliente.

Componentes



Um componente guarda a programação necessária para “desenhar” o HTML, que será enviado para a página única.

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">…
      </header>
    </div>
  );
}

export default App;
```

```
import logo from './logo.svg';
import './App.css';
import { Component } from 'react';

class App extends Component {
  render() {
    return (
    <div className="App">
      <header className="App-header">…
      </header>
    </div>
    )
  }
}

export default App;
```

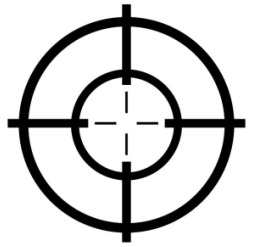
Propriedades

Conceitualmente, componentes são funções, que aceitam entradas arbitrárias chamadas de “props” – props é uma abreviação de properties.

Props é um objeto que armazena uma serie de dados, estes dados são utilizados na renderização do componente. Através da variável props se passa os dados para a renderização ocorrer.

Importante! O nome props é uma convenção, comum entre os desenvolvedores JavaScript.

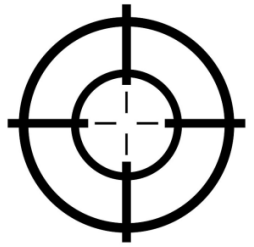
Propriedades



```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
ReactDOM.render(
  <React.StrictMode>

    <App mensagem = 'Texto enviado como propriedade' />

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

```jsx
import './App.css';

function App(props) {
  return (
    <div className="App">
      <header className="App-header">
        <p>
          {props.mensagem}
        </p>
      </header>
    </div>
  );
}
export default App;
```

Propriedades



```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
ReactDOM.render(
  <React.StrictMode>

    <App titulo='Título enviado' texto='texto enviado' />

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

```jsx
import './App.css';

function App(props) {
  return (
    <div className="App">
      <header className="App-header">
        <h1>
          {props.titulo}
        </h1>
        <p>
          {props.texto}
        </p>
      </header>
    </div>
  );
}
export default App;
```

Props é um objeto, que armazena a serie de propriedades, com seus respectivos valores!

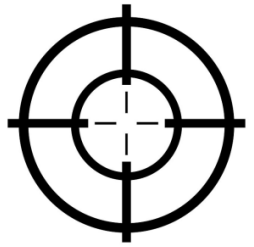
Composição de componentes

Um componente pode ser uma parte reutilizável!

```
function Info(props) {
    return (
        <div>
            <h1>
                {props.titulo}
            </h1>
            <p>
                {props.texto}
            </p>
            <br />
        </div>
    )
}
export default Info;
```

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

import Info from './info';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Info titulo = 'Título 1' texto = 'Texto 1' />
        <Info titulo = 'Título 2' texto = 'Texto 2' />
        <Info titulo = 'Título 3' texto = 'Texto 3' />
      </header>
    </div>
  );
}
export default App;
```

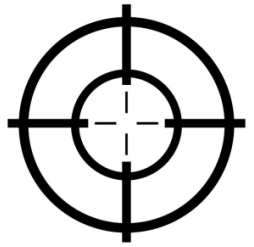
Propriedades em classes



```
class Info extends Component<{ titulo: string, texto: string }> {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>{this.props.titulo}</h1>
        <p>{this.props.texto}</p>
      </div>
    );
  }
}
export default Info
```

```
import Info from './info';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Info titulo = 'Título 1' texto = 'Texto 1' />
        <Info titulo = 'Título 2' texto = 'Texto 2' />
        <Info titulo = 'Título 3' texto = 'Texto 3' />
      </header>
    </div>
  );
}
export default App;
```

Propriedades em classes



```
type props = {
    titulo: string,
    texto: string
}
class Info extends Component<props> {
    constructor(props) {
        super(props);
    }
    render() {
        return (
            <div>
                <h1>{this.props.titulo}</h1>
                <p>{this.props.texto}</p>
            </div>
        );
    }
}
export default Info
```

Objeto que representa a estrutura das propriedades!

Importante! Quando se utiliza um componente de classe é necessário definir como será a estrutura, os nomes e os tipos dos dados passados como propriedades.

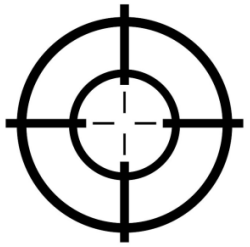
Propriedades em classes

https://github.com/gerson-pn

/react-props-class

/react-props-type-class

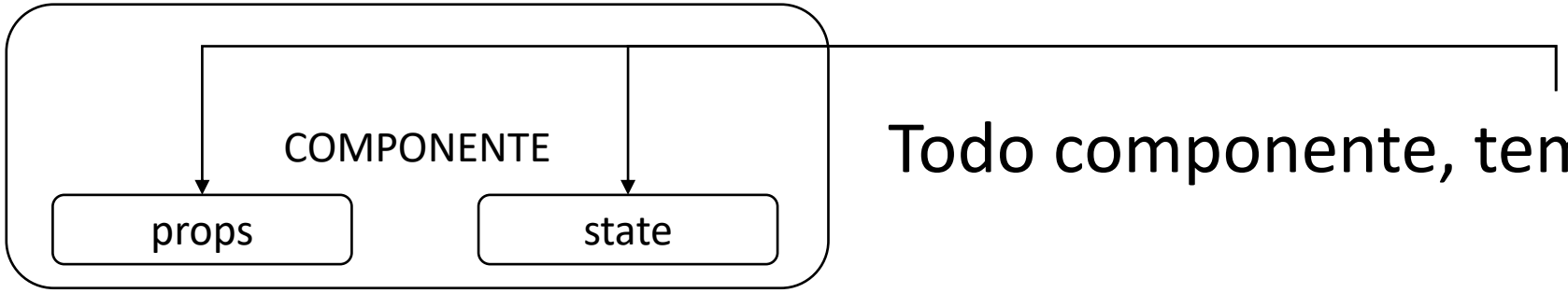
Estado e ciclo de vida



A documentação da biblioteca react designa as propriedades de um componente como do tipo somente leitura, para o próprio componente. Portanto, um componente não deve alterar as props.

Para alterar os atributos de um componente utiliza-se o conceito de estado (state). O state permite dinamismo no momento da renderização do componente, ou seja, alterar sua saída ao longo do tempo em resposta a ações do cliente ou por outra necessidade.

Estado e ciclo de vida



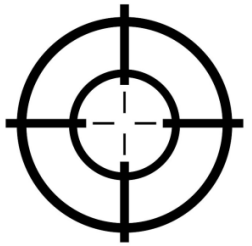
Todo componente, tem props e state.

As props são valores passados para construção do componente.

O state é um objeto, interno, do componente. Ele serve para armazenar valores ou informações que representam o estado do componente.

O state pode ser modificado, mas não diretamente, para isso utiliza-se a função interna chamada setState().

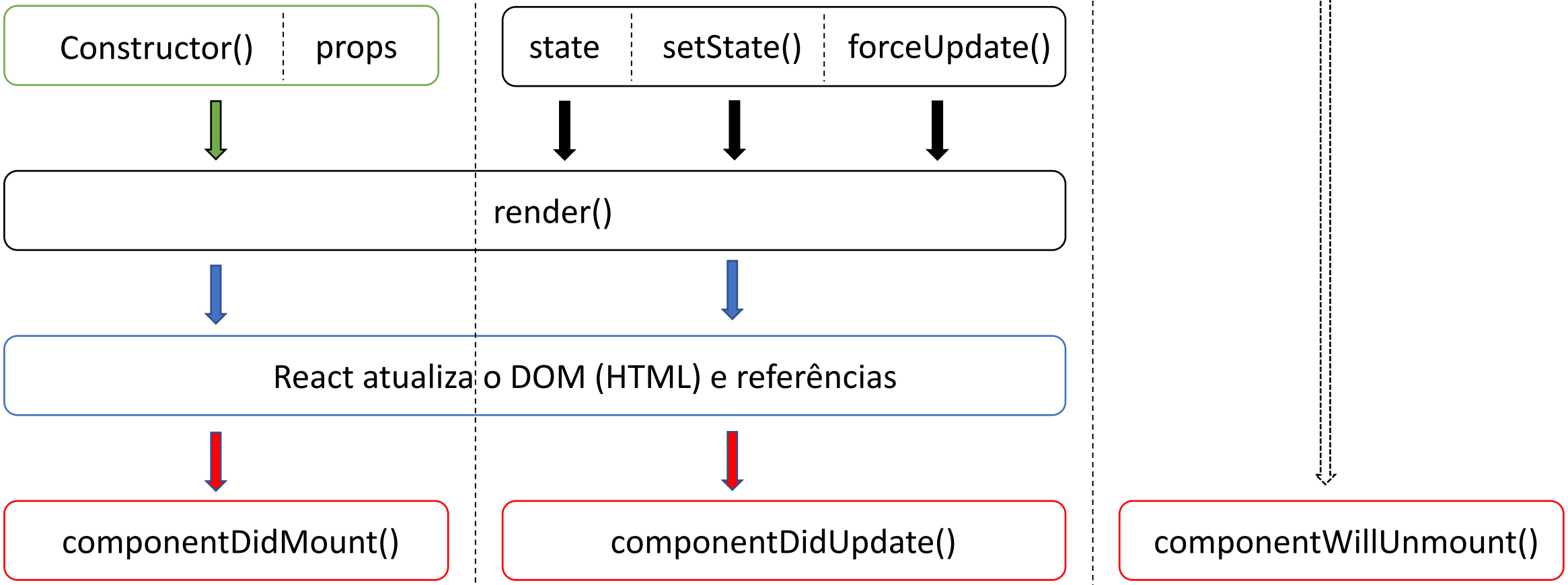
Estado e ciclo de vida



1 Montagem

2 Atualização

3 Desmontagem



Estado e ciclo de vida

componentDidMount() é invocado imediatamente após um componente ser montado.

componentDidUpdate() é invocado imediatamente após alguma atualização ocorrer, uma modificação do state.

componentWillUnmount() é invocado antes que um componente seja desmontado e destruído.

Estado e ciclo de vida

Importante! Props é um objeto, que serve para passar dados ao componente! Props é do tipo somente leitura!

Dentro de um componente existe outro objeto, que mantem seu estado, o state. O state armazena os dados que definem o estado do componente e que podem ser alterados durante o ciclo de vida.

Importante! Em componentes de classe, a estrutura do state precisa ser definida, assim como no props.

```
type state = {
    data: Date
}
class Componente extends Component<{}, state> {
    constructor(props) {
        super(props);
        this.state = {
            data: new Date()
        }
    }
    render() {
        return (
            <div>
                <h1>Data e hora de agora</h1>
                <h2>
                Data: {this.state.data.toLocaleDateString()}
                <span> </span>
                Hora: {this.state.data.toLocaleTimeString()}
                </h2>
            </div>
        );
    }
}
export default Componente
```

Estrutura do objeto state!

Estrutura do objeto props, se existir!

Inicialização do estado do componente!

A inicialização ocorre sempre no construtor, antes de ocorrer a renderização!

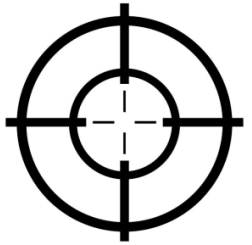
```
class Componente extends Component<{}, { data: Date }> {
    private temporizador
    constructor(props) {···
    }

    alteracao() {
        this.setState({
            data: new Date()
        })
    }

    componentDidMount() {
        this.temporizador = setInterval(
            () => this.alteracao(),
            1000
        );
    }

    render() {···
    }
}
export default Componente
```

Importante! O state é do tipo somente leitura, como o props. Por isso, utiliza-se o método setState() para atualizar seus dados, fora do método construtor!

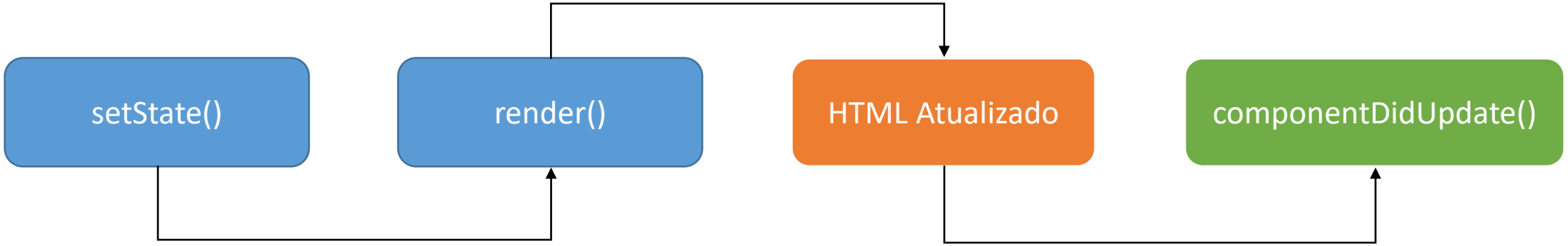
Obrigatoriamente, o método recebe um objeto, que deve conter atributos de mesmo nome e tipo como definido no state, este objeto é o novo estado!

Toda vez que os dados do state são alterados, o componente se autoatualiza.

Estado e ciclo de vida



Lembre-se! Sempre após a invocação do método setState() a renderização do componente é refeita, ou seja, o método render é executado novamente e o valor do state é reutilizado! Todo o componente é montado novamente e o HTML é atualizado!



Estado e ciclo de vida

https://github.com/gerson-pn

/react-state-type-class

/react-dynamic-state-type-class

```
class Componente extends Component<{ texto: string }, { data: Date, texto: string }> {
    private temporizador
    constructor(props) {
        super(props);
        this.state = {
            data: new Date(),
            texto: this.props.texto
        }
    }
    alteracao() {
        this.setState({
            data: new Date()
        })
    }
    componentDidMount() {
        this.temporizador = setInterval(
            () => this.alteracao(),
            1000
        );
    }
    render() {
        return (
            <div>
                <h1>{this.state.texto}</h1>
                <h2>
                    Data: {this.state.data.toLocaleDateString()}
                    <span> </span>
                    Hora: {this.state.data.toLocaleTimeString()}
                </h2>
```

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Componente texto='Data e hora de agora'/>
      </header>
    </div>
  );
}
```

Pode-se utilizar props para preencher dados no state. Contudo recomenda-se não fazer isso diretamente, exceto se o valor for mudar dinamicamente durante o ciclo de vida.

```
ReactDOM.render(
  <React.StrictMode>
    <App texto = 'Data e hora de agora' />
  </React.StrictMode>,
  document.getElementById('root')
);
```

```
function App(props) {
  return (
    <div className="App">
      <header className="App-header">
        <Componente texto={props.texto} />
      </header>
    </div>
  );
}
```

```
class Componente extends Component<{ texto: string }, { data: Date, texto: string }> {
    private temporizador
    constructor(props) {…
    }
    alteracao() {…
    }
    componentDidMount() {…
    }
    render() {…
    }
}
```

As propriedades podem ser repassadas pela hierarquia de componentes, o fluxo é sempre do componente pai para o componente filho.

Estado e ciclo de vida

https://github.com/gerson-pn

/react-dynamic-state-props-type-class

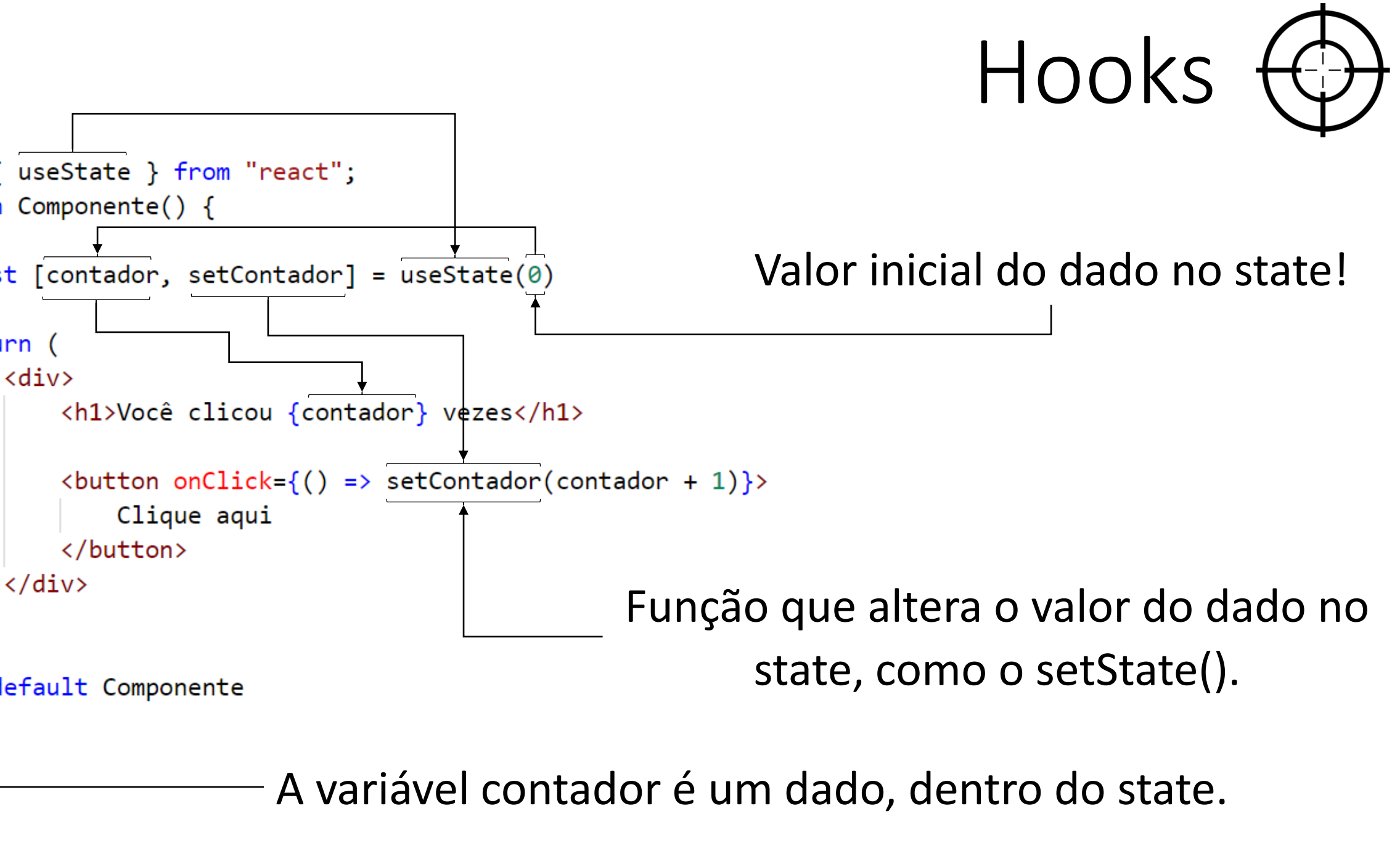
/react-passing-properties

Hooks

Ganchos (hooks) são funções e uma alternativa para construção de componentes sem utilizar classes. Foram introduzidos na versão 16.8 da biblioteca para agilizar o uso do state e de outros recursos.

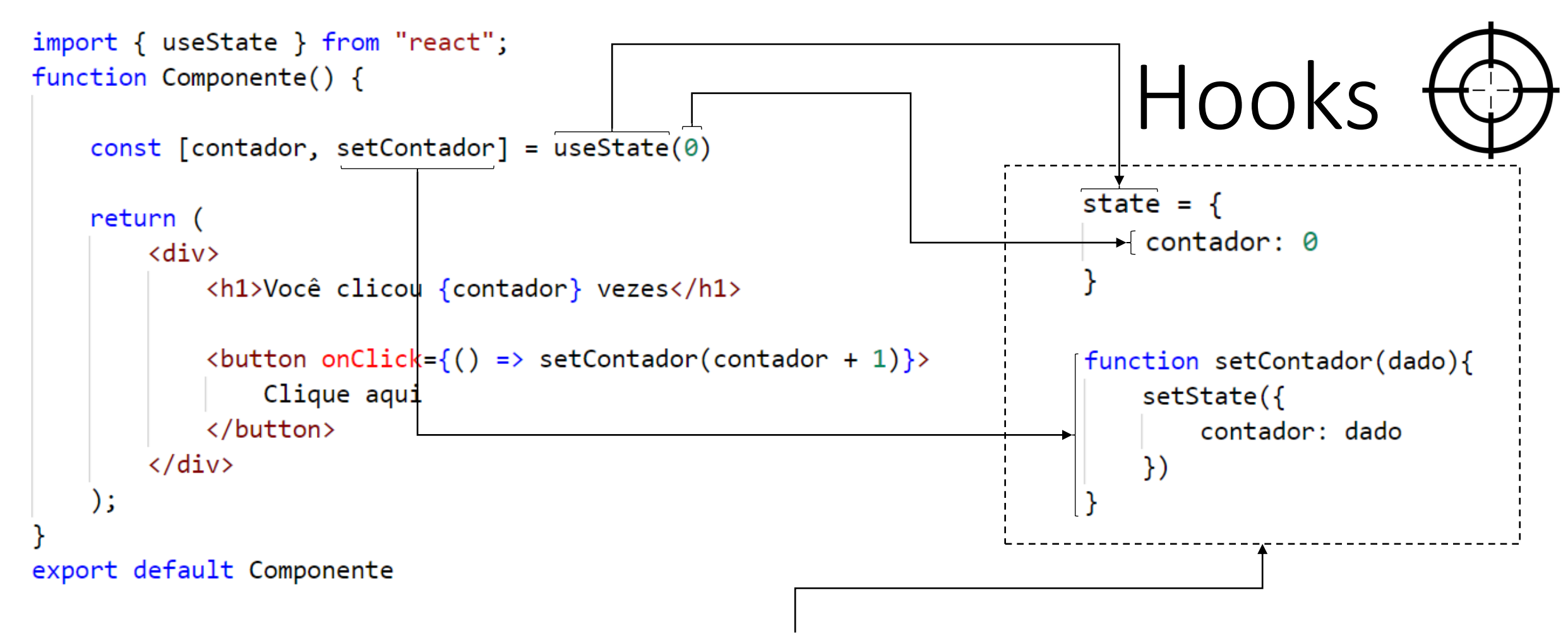
Apesar de hooks agilizarem o desenvolvimento, de acordo com a documentação da biblioteca, não há planos de excluir classes. Portanto, cabe ao desenvolvedor escolher qual é a melhor estratégia para o contexto do seu desenvolvimento.

Hooks



```
import { useState } from "react";
function Componente() {

    const [contador, setContador] = useState(0)

    return (
        <div>
            <h1>Você clicou {contador} vezes</h1>

            <button onClick={() => setContador(contador + 1)}>
                Clique aqui
            </button>
        </div>
    );
}
export default Componente
```

Hooks



```
import { useState } from "react";
function Componente() {

    const [contador, setContador] = useState(0)

    return (
        <div>
            <h1>Você clicou {contador} vezes</h1>

            <button onClick={() => setContador(contador + 1)}>
                Clique aqui
            </button>
        </div>
    );
}
export default Componente
```

```
state = {
    contador: 0
}

function setContador(dado){
    setState({
        contador: dado
    })
}
```

Na prática, o useState() cria um state e preenche o valor do dado, dentro do state. Também entrega uma função, que encapsula a chamada do setState(). Tudo isto é implícito, feito por “debaixo dos panos” automaticamente.

Comparação entre classe e função.

```tsx
class Componente extends Component<{}, { contador: number }> {
    constructor(props) {
        super(props)

        this.state = {
            contador: 0
        }
    }
    render() {
        return (
            <div>
                <h1>Você clicou {this.state.contador} vezes</h1>
                <button onClick={() => this.setState({ contador: this.state.contador + 1 })}>
                    Clique aqui
                </button>
            </div>
        );
    }
}
export default Componente
```

```tsx
import { useState } from "react";
function Componente() {

    const [contador, setContador] = useState(0)

    return (
        <div>
            <h1>Você clicou {contador} vezes</h1>

            <button onClick={() => setContador(contador + 1)}>
                Clique aqui
            </button>
        </div>
    );
}
export default Componente
```

Hooks

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

function Componente(props) {
    const [data, setData] = useState(new Date())

    useEffect(() => {
        const temporizador = setInterval(() =>
            setData(new Date()),
            1000
        )
    })
    return (
        <div>
            <h1>{props.texto}</h1>
            <h2>
                Data: {data.toLocaleDateString()}
                <span> </span>
                Hora: {data.toLocaleTimeString()}
            </h2>
        </div>
    )
}
export default Componente
```

A função useEffect() é um “hook”, que realiza alguma lógica para atualizar o componente durante o ciclo de vida, similar ao componentDidMount() e componentDidUpdate(). Na prática, a chamada desta função corresponde aos métodos combinados.

```
class Componente extends Component<{ texto: string }, { data: Date, texto: string }> {
    constructor(props) {
        super(props);
        this.state = {
            data: new Date(),
            texto: this.props.texto
        }
    }
    componentDidMount() {
        setInterval(
            () => this.setState({
             data: new Date()
            }),
            1000);
    }
    render() {
        return (
            <div><h1>{this.state.texto}</h1>
                <h2>Data: {this.state.data.toLocaleDateString()}
                <span> </span>
                Hora: {this.state.data.toLocaleTimeString()}
                </h2>
            </div>
        );
    }
}
export default Componente
```

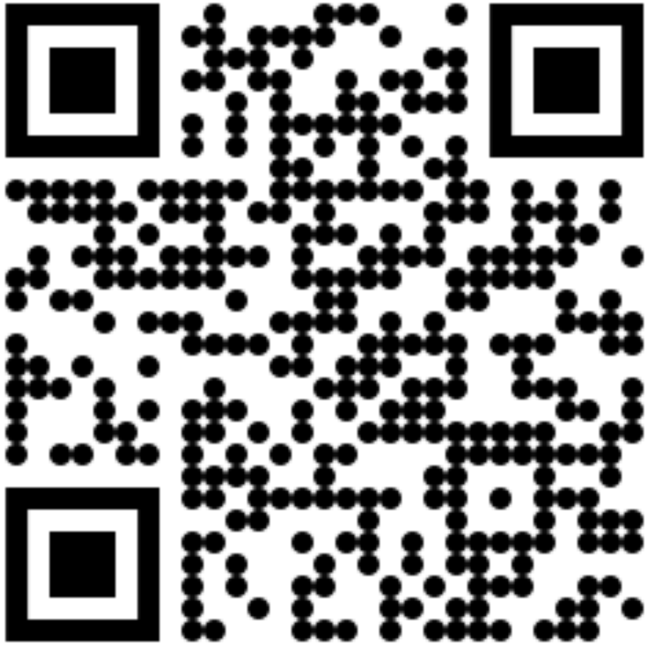
```
function Componente(props) {
    const [data, setData] = useState(new Date())
    useEffect(() => {
        setInterval(() =>
            setData(new Date()),
            1000
        )
    })
    return (
        <div>
            <h1>{props.texto}</h1>
            <h2>Data: {data.toLocaleDateString()}
            <span> </span>
            Hora: {data.toLocaleTimeString()}
            </h2>
        </div>
    )
}
export default Componente
```

Hooks

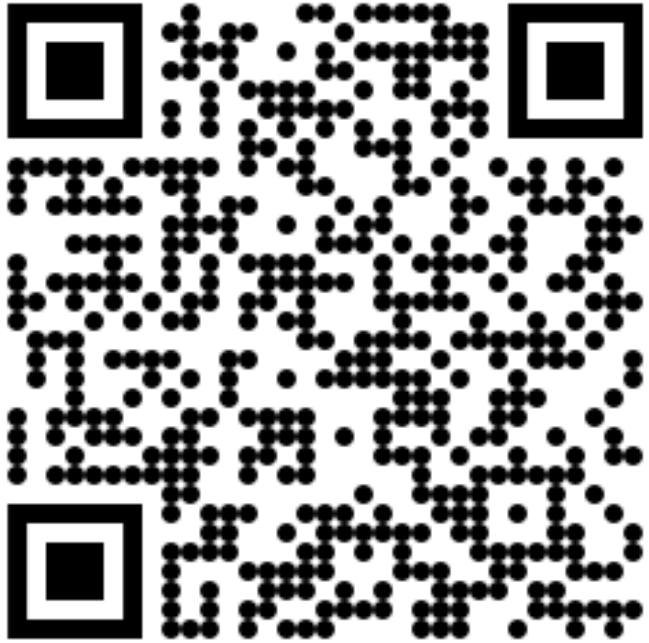
Hooks

https://github.com/gerson-pn

/react-hook-counter



/react-hook-effect-timer



Hooks

Existem algumas motivações para utilizar hooks, mas o principal é:

A documentação da biblioteca entende que classes podem ser uma grande barreira no aprendizado da biblioteca e impactar no seu uso futuro por desenvolvedores iniciantes ou que não tenham familiaridade com programação orientada à objetos.

Atualmente a documentação encoraja a utilizar hooks, mas hooks não cobrem todas as possibilidades do ciclo de vida. Deve-se lembrar que TypeScript não depende da biblioteca para continuar em uso. Portanto o conhecimento de programação orientada a objetos ainda é fundamental.