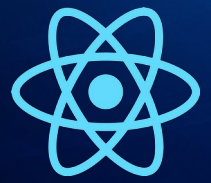
DATAPREV



React

UMA BIBLIOTECA JAVASCRIPT PARA CRIAR INTERFACES DE USUÁRIO

Ricardo Glodzinski
Analista de Tecnologia da Informação

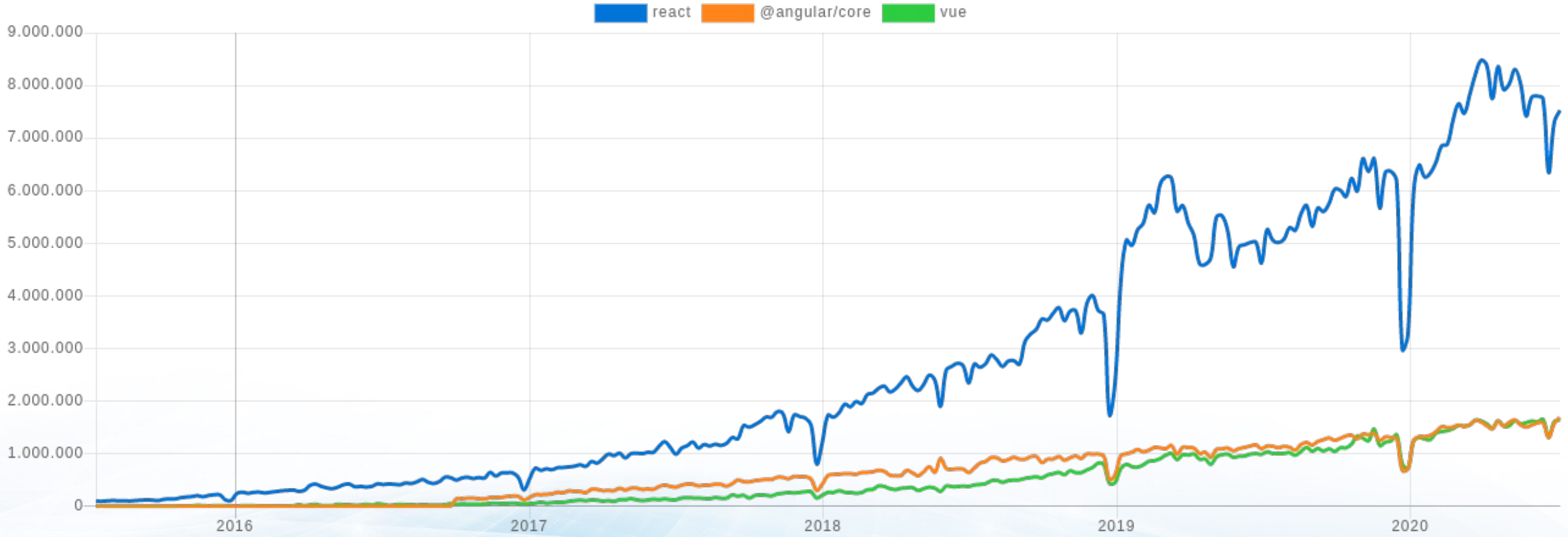
AULA 1

PLANEJAMENTO

- Sobre o React
- Montagem do ambiente de desenvolvimento
- Iniciando uma aplicação React
- JSX
- Renderizando elementos
- Componentes e Props
- Estado e ciclo de vida
- Manipulação de eventos
- Renderização condicional
- Atividades assíncronas: exercícios

SOBRE O REACT

EVOLUÇÃO DE DOWNLOADS – ÚLTIMOS 5 ANOS



Fonte:
https://www.npmtrends.com/react-vs-@angular/core-vs-vue
Data: 16/07/2020

SOBRE O REACT

UMA BIBLIOTECA JAVASCRIPT PARA CRIAR INTERFACES DE USUÁRIO

- Lançado em 29 de maio de 2013 (7 anos).
- Software de código aberto.
- Desenvolvido inicialmente pelo Facebook.
- Declarativo.
- Baseado em componentes.
- É uma biblioteca e não um framework.

SOBRE O REACT

LINGUAGEM DECLARATIVA

- É, geralmente, o contrário de linguagem imperativa.
- A declarativa se preocupa com o que o programador quer fazer, a imperativa quer saber como atingir o objetivo desejado.

```
// IMPERATIVA
if(usuario.curtiu()) {
  if(botaoEstaAzul()) {
     removeAzul();
     adicionaCinza();
  } else {
     removeCinza();
     adicionaAzul();
  }
}
```

```
// DECLARATIVA

if(this.state.curtido) {
   return <CurtidaAzul />;
} else {
   return <CurtidaCinza /> ;
}
```

AMBIENTE DE DESENVOLVIMENTO LOCAL

FERRAMENTAS QUE SERÃO UTILIZADAS NO CURSO

GIT

Sistema de controle de versão

https://git-scm.com/

Visual Studio Code

IDE para edição de código

https://code.visualstudio.com/

Node.js®

Ambiente que permite execução de JavaScript do lado do servidor.

Utilizar o NVM (Node Version Manager) para instalar

https://github.com/nvm-sh/nvm

Hands on!

AMBIENTE DE DESENVOLVIMENTO LOCAL

INICIANDO UMA APLICAÇÃO REACT

Toolchains recomendadas

- Nova aplicação SPA: Create React App
- Site renderizado no servidor (SSR) com Node.js: Next.js
- Site estático orientado a conteúdo: Gatsby
- Biblioteca de componentes
  - Neutrino
  - Nx
  - Parcel
  - Razzle

AMBIENTE DE DESENVOLVIMENTO LOCAL

INICIANDO UMA APLICAÇÃO REACT

Iremos utilizar o create react app:

```
npx create-react-app nome-da-app
cd nome-da-app
npm start
```

Hands on!

OLÁ MUNDO!

MENOR EXEMPLO DE REACT

```
ReactDOM.render(
  <h1>Olá, mundo!</h1>,
  document.getElementById('root')
);
```

Hands on!

JSX

UMA EXTENSÃO DE SINTAXE PARA JAVASCRIPT

```
const element = <h1>Olá, mundo!</h1>;
```

- Não é uma string, nem HTML
- Serve para descrever como a UI deveria parecer
- Produz “elementos” do React
- NÃO é interpretado pelo navegador

JSX

INCORPORANDO EXPRESSÕES

```
const nome = 'Ricardo Glodzinski';
const elemento = <h1>Olá, {nome}</h1>;

ReactDOM.render(
  elemento,
  document.getElementById('root')
);
```

É possível inserir qualquer expressão JavaScript válida dentro das chaves em JSX. Por exemplo, 2 + 2, user.firstName, ou formatName(user) são todas expressões JavaScript válidas.

JSX

ATRIBUTOS

```
// Aspas para string
const element = <div tabIndex="0"></div>;

// Chaves para expressões
const element = <img src={user.avatarUrl}></img>;
```

Usa camelCase como convenção para nomes de propriedades ao invés dos nomes de atributos do HTML.

Por exemplo, class se transforma em className e tabindex se transforma em tabIndex.

JSX

ELEMENTOS FILHOS

```
// Se não tiver um elemento filho pode ser fechada com />
const element = <img src={user.avatarUrl} />;

// Elemento com filhos
const element = (
  <div>
    <h1>Olá!</h1>
    <h2>É bom ver você aqui!</h2>
  </div>
);
```

JSX

REPRESENTA OBJETOS

```
// Estes dois exemplos são idênticos:
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);

// O Babel compila JSX para chamadas React.createElement()
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

RENDERIZANDO ELEMENTOS

```
<div id="root"></div>
```

```
const element = <h1>Olá, mundo!</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

- O root, nesse caso, é o nó raiz do DOM. Tudo dentro dele será gerenciado pelo React DOM.
- Para renderizar algo dentro do “root”, chamar a função ReactDOM.render(), passando o elemento é o nó do DOM como parâmetros.
- Aplicações React, geralmente, possuem um único nó raiz.

RENDERIZANDO ELEMENTOS

IMUTABILIDADE

```
function tick() {
  const elemento = (
    <div>
      <h1>Olá, mundo!</h1>
      <h2>Agora é {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(elemento, document.getElementById('root'));
}

setInterval(tick, 1000);
```

Hands on!

- Elementos React são imutáveis. Uma vez criados, você não pode alterar seus elementos filhos ou atributos.
- O React Somente Atualiza o Necessário

COMPONENTES E PROPS

- UI dividida em partes independentes e reutilizáveis.
- Componentes são como funções JS. Eles recebem entradas arbitrárias, chamadas de props, e retornam elementos React.
- Existem dois tipos de componentes: componentes de função e componentes de classe.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

RENDERIZANDO COMPONENTES

```
function Ola(props) {
  return <h1>Olá, {props.nome}</h1>;
}

const elemento = <Ola nome="Ricardo" />;

ReactDOM.render(
  elemento,
  document.getElementById('componente-ola')
);
```

- Quando o React vê um elemento representando um componente definido pelo usuário, ele passa atributos JSX e componentes filhos para esse componente como um único objeto. Nós chamamos esse objeto de “props”.

NOTA: Sempre inicie os nomes dos componentes com uma letra maiúscula.

COMPOSIÇÃO DE COMPONENTES

```
function Ola(props) {
  return <h1>Olá, {props.nome}</h1>;
}

function Composicao() {
  return (
    <div>
      <Ola nome="Ricardo" />
      <Ola nome="João" />
    </div>
  );
}

ReactDOM.render(
  <Composicao />,
  document.getElementById('composicao')
);
```

EXTRAINDO COMPONENTES

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

```
function Avatar(props) {
  return (
    <img className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />
  );
}
```

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

EXTRAINDO COMPONENTES

```
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

```
function Avatar(props) {
  return (
    <img className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />
  );
}
```

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

Hands on!

PROPS

- Independente se você declarar um componente como uma função ou uma classe, ele nunca deve modificar seus próprios props.
- Todos os componentes React tem que agir como funções puras em relação ao seus props.

```
function sum(a, b) {
  return a + b;
};
```

FUNÇÃO PURA

```
function withdraw(account, amount) {
  account.total -= amount;
};
```

FUNÇÃO IMPURA

ESTADO

- O componente precisa ser uma classe ES6 estendendo React.component.

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

class Clock extends Component {
    render() {
        return (
            <div>
            </div>
        );
    }
}

export default Clock;
```

ESTADO

- O estado é um objeto armazenado em this.state e é inicializado no construtor da classe.

```
class Clock extends Component {
    constructor(props) {
        super(props);
        this.state = {};
    }

    render() {
        return (
            <div>
            </div>
        );
    }
}

export default Clock;
```

ESTADO

- Para manipular os valores do estado do componente deve ser usada a função this.setState(). O estado não deve ser manipulado diretamente.

- As atualizações do estado podem ser assíncronas.

```
// Errado
this.setState({
  counter: this.state.counter + this.props.increment,
});

// Correto
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

ESTADO

- O React mescla o objeto passado como parâmetro para this.setState com os valores já contidos no estado. É superficial.

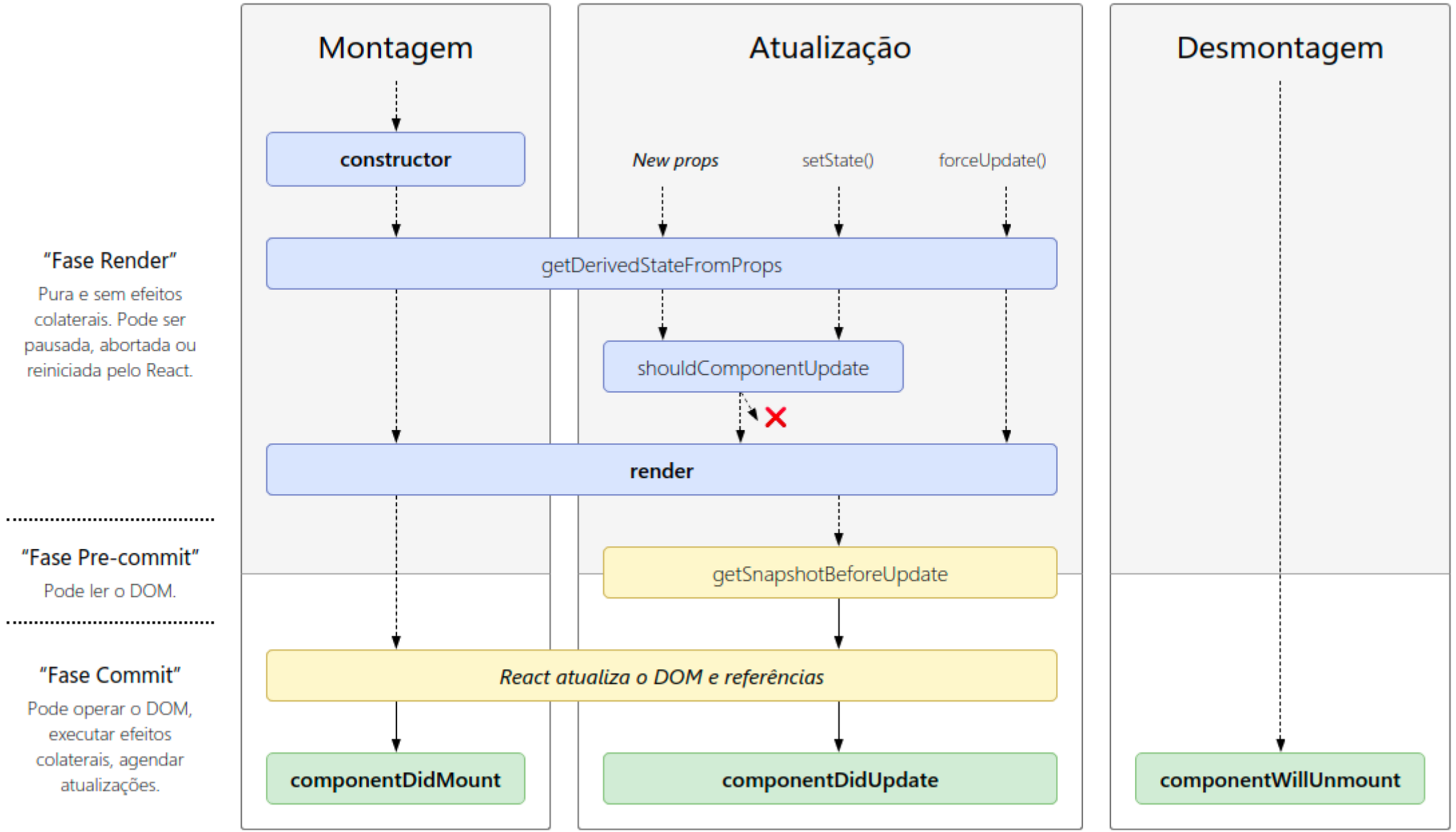
```
constructor(props) {
  super(props);
  this.state = {
    posts: [],
    comments: []
  };
}

componentDidMount() {
  fetchPosts().then(response => {
    this.setState({
      posts: response.posts
    });
  });
}
```

ESTADO

- Os dados fluem para baixo.
- O estado é geralmente chamado de local ou encapsulado. Não é acessível a nenhum componente que não seja o que o possui e o define.
- Um componente pode escolher passar seu estado como props para seus componentes filhos.
- Nos apps React, se um componente é stateful ou stateless é considerado um detalhe de implementação do componente que pode mudar com o tempo. Você pode usar componentes sem estado dentro de componentes com estado e vice-versa.

CICLO DE VIDA



COMPONENTE COMPLETO

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

class Clock extends Component {
    constructor(props) {
        super(props);
        this.state = { date: new Date() };
    }

    componentDidMount() {
        this.timerID = setInterval(
            () => this.tick(),
            1000
        );
    }

    componentWillUnmount() {
        clearInterval(this.timerID);
    }

    tick() {
        this.setState({
            date: new Date()
        });
    }

    render() {
        return (
            <div>
                <h2>Agora é {this.state.date.toLocaleTimeString()}.</h2>
            </div>
        );
    }
}

export default Clock;
```

Hands on!

MANIPULAÇÃO DE EVENTOS

- Eventos em React são nomeados usando camelCase.
- É passada uma função para manipular o evento.

HTML

```
<button onclick="activateLasers()">
  Ativar lasers
</button>
```

React

```
<button onClick={activateLasers}>
  Ativar lasers
</button>
```

MANIPULAÇÃO DE EVENTOS

- Não se pode retornar false para evitar o comportamento padrão no React, como é no HTML. É necessário chamar preventDefault explicitamente

HTML

```
<a href="#" onclick="console.log('O link foi clicado.'); return false">
  Clique Aqui
</a>
```

React

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('O link foi clicado.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Clique Aqui
    </a>
  );
}
```

MANIPULAÇÃO DE EVENTOS

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

class ToggleButton extends Component {

    constructor(props) {
        super(props);
        this.state = { isToggleOn: false };

        // Aqui utilizamos o `bind` para que o `this` funcione dentro da nossa callback
        this.handleClick = this.handleClick.bind(this);
    }

    handleClick() {
        this.setState(state => ({
            isToggleOn: !state.isToggleOn
        }));
    }

    render() {
        return (
            <button onClick={this.handleClick}>
                {this.state.isToggleOn ? 'ON' : 'OFF'}
            </button>
        );
    }
}

export default ToggleButton;
```

Hands on!

MANIPULAÇÃO DE EVENTOS

CUIDADOS COM O “this”

- Em JavaScript, os métodos de classe não são vinculados por padrão. Se não for feito o bind e passá-lo para um onClick, o this será undefined quando a função for realmente chamada.
- Este não é um comportamento específico do React.
- É possível contornar essa situação e não ter que fazer o bind de duas maneiras.
    - Usando a sintaxe experimental de campos de classe pública.
    - Usando uma arrow function como callback.

MANIPULAÇÃO DE EVENTOS

CUIDADOS COM O “this”

Campos de classe pública

```
class LoggingButton extends React.Component {
  handleClick = () => {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Clique Aqui
      </button>
    );
  }
}
```

Arrow function

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={() => this.handleClick()}>
        Click me
      </button>
    );
  }
}
```

* O problema com esta sintaxe é que um callback diferente é criado toda vez que o LoggingButton é renderizado. Na maioria dos casos, tudo bem. No entanto, se esse callback for passado para componentes inferiores através de props, esses componentes poderão fazer uma renderização extra. Geralmente recomendamos a vinculação no construtor ou a sintaxe dos campos de classe para evitar esse tipo de problema de desempenho.

MANIPULAÇÃO DE EVENTOS

PASSANDO ARGUMENTOS

```
<button onClick={(e) => this.deleteRow(id, e)}>Deletar linha</button>

<button onClick={this.deleteRow.bind(this, id)}>Deletar linha</button>
```

- As duas linhas acima são equivalentes e usam arrow function e Function.prototype.bind respectivamente.
- Em ambos os casos, o argumento e representando o evento do React será passado como segundo argumento após o ID.

RENDERIZAÇÃO CONDICIONAL

- Renderização condicional em React funciona da mesma forma que condições funcionam em JavaScript.

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

RENDERIZAÇÃO CONDICIONAL

POSSÍVEIS CONDICIONAIS

- If inline com o Operador Lógico &&

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}
```

RENDERIZAÇÃO CONDICIONAL

POSSÍVEIS CONDICIONAIS

- If-Else inline com Operador Condicional

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn
        ? <LogoutButton onClick={this.handleLogoutClick} />
        : <LoginButton onClick={this.handleLoginClick} />
      }
    </div>
  );
}
```

RENDERIZAÇÃO CONDICIONAL

POSSÍVEIS CONDICIONAIS

- Evitando que um Componente seja Renderizado

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}
```

NOTA: Retornar null do método render de um componente não afeta a ativação dos métodos do ciclo de vida do componente.

RENDERIZAÇÃO CONDICIONAL

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

function UserGreeting(props) {
    return <h1>Você está logado!</h1>;
};

function GuestGreeting(props) {
    return <h1>Você não está logado!</h1>;
};

function Greeting(props) {
    const isLoggedIn = props.isLoggedIn;
    if (isLoggedIn) {
        return <UserGreeting />;
    }
    return <GuestGreeting />;
};
```

Hands on!

```
class LoginControl extends Component {

    constructor(props) {
        super(props);
        this.handleLoginLogout = this.handleLoginLogout.bind(this);
        this.state = { isLoggedIn: false };
    }

    handleLoginLogout() {
        this.setState(state => ({ isLoggedIn: !state.isLoggedIn }));
    }

    render() {
        const isLoggedIn = this.state.isLoggedIn;

        let loginLogoutBtn = (
            <button onClick={this.handleLoginLogout}>
                {isLoggedIn ? 'Sair' : 'Entrar'}
            </button>
        );

        return (
            <div>
                <Greeting isLoggedIn={isLoggedIn} />
                {loginLogoutBtn}
            </div>
        );
    }
}

export default LoginControl;
```

Obrigado!

Ricardo Glodzinski
Analista de Tecnologia da Informação
ricardo.glodzinski@dataprev.gov.br

Julho 2020

www.facebook.com/dataprevtecnologia

@dataprev

DATAPREV