



**DATAPREV**



UMA BIBLIOTECA JAVASCRIPT PARA CRIAR INTERFACES DE USUÁRIO

---

Ricardo Glodzinski  
Analista de Tecnologia da Informação

Tiago de Andrade Freire  
Analista de Tecnologia da Informação







# AULA 4

## PLANEJAMENTO

- Refs
- Acessibilidade
- Divisão de código
- Contexto
- Error Boundaries
- Componentes de ordem superior (HOCs)
- Portals
- Checagem de tipos com PropTypes
- Atividades assíncronas: exercícios

# REFS

- Refs fornecem uma forma de acessar os nós do DOM ou elementos React criados no método render.
- Em um fluxo de dados típico do React, as props são a única forma de componentes pais interagirem com seus filhos.
- Existem alguns casos onde você precisa modificar imperativamente um componente filho fora do fluxo típico de dados. O componente filho a ser modificado poderia ser uma instância de um componente React, ou ele poderia ser um elemento DOM.

- Gerenciamento de foco, seleção de texto, ou reprodução de mídia.
- Engatilhar animações imperativas.
- Integração com bibliotecas DOM de terceiros.
- Evite usar refs para qualquer coisa que possa ser feita de forma declarativa.
- Por exemplo, ao invés de expôr os métodos `open()` e `close()` em um componente `Dialog`, passe a propriedade `isOpen` para ele.

# REFS

CRIANDO

- Refs são criadas usando `React.createRef()` e anexadas aos elementos React por meio do atributo `ref`.
- As Refs são comumente atribuídas a uma propriedade de instância quando um componente é construído para que então elas possam ser referenciadas por todo o componente.

```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);  
    this.myRef = React.createRef();  
  }  
  render() {  
    return <div ref={this.myRef} />;  
  }  
}
```



# REFS

CRIANDO

- Quando uma ref é passada para um elemento no render, uma referência para o nó se torna acessível no atributo current da ref.
- Quando o atributo ref é usado em um elemento HTML, a ref criada no construtor `React.createRef()` recebe um elemento DOM subjacente como a propriedade current.
- Quando o atributo ref é usado em um componente de classe, o objeto ref recebe uma instância montada de um componente em sua propriedade current.
- Você não pode usar o atributo ref em um componente funcional, já que eles não possuem instâncias.

# REFS

NÓ DO DOM

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

class CustomTextInput extends Component {

  constructor(props) {
    super(props);
    this.textInput = React.createRef();
    this.focusTextInput = this.focusTextInput.bind(this);
  }

  focusTextInput() {
    this.textInput.current.focus();
  }

  render() {
    return (
      <div>
        <input type="text" ref={this.textInput} />
        <input
          type="button"
          value="Focar no input de texto" onClick={this.focusTextInput} />
      </div>
    );
  }
}

export default CustomTextInput;
```



Hands on!



# REFS

## COMPONENTE DE CLASSE

```
import React, { Component } from "react";
import CustomTextInput from "../CustomTextInput";

class AutoFocusTextInput extends Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }

  componentDidMount() {
    this.textInput.current.focusTextInput();
  }

  render() {
    return (
      <CustomTextInput ref={this.textInput} />
    );
  }
}

export default AutoFocusTextInput;
```



Hands on!

# ACESSIBILIDADE

- A acessibilidade da Web é o design e a criação de sites que podem ser usados por todos.
- O suporte à acessibilidade é necessário para permitir que tecnologias assistivas interpretem as páginas da web.
- React suporta totalmente a construção de sites acessíveis, muitas vezes usando técnicas HTML padrão.

# ACESSIBILIDADE

WCAG – WEB CONTENT ACCESSIBILITY GUIDELINES

- O Web Content Accessibility Guidelines fornece diretrizes para a criação de sites acessíveis.
- As seguintes checklists das WCAG fornecem uma visão geral:
  - [WCAG checklist from Wuhcag](#)
  - [WCAG checklist from WebAIM](#)
  - [Checklist from The A11Y Project](#)

# ACESSIBILIDADE

WAI-ARIA – WEB ACCESSIBILITY INITIATIVE - ACCESSIBLE RICH INTERNET APPLICATIONS

- O [documento WAI-ARIA](#) contém técnicas para a criação de widgets JavaScript totalmente acessíveis.
- Todos os atributos HTML aria-\* são totalmente suportados no JSX.
- Enquanto a maioria das propriedades e atributos do DOM no React são camelCase, esses atributos devem ser hyphen-case.

```
<input  
  type="text"  
  aria-label={labelText}  
  aria-required="true"  
  onChange={onchangeHandler}  
  value={inputValue}  
  name="name"  
>
```



# ACESSIBILIDADE

HTML

- Linguagem é a base da acessibilidade em um aplicativo da web.
- Usando corretamente os elementos HTML para reforçar o significado da informação em nossos sites, muitas vezes a acessibilidade vem gratuitamente.
- É importante não quebrar a **semântica** dos elementos.
- Às vezes, é quebrada a semântica de HTML ao adicionar elementos `<div>` ao JSX somente para fazer o código React funcionar, especialmente ao trabalhar com listas (`<ol>`, `<ul>` e `<dl>`) e HTML `<table>`. Nesses casos, deve ser utilizado **React Fragments** para agrupar vários elementos.

# ACESSIBILIDADE

HTML

```
import React from "react";

const ListaItem = ({ item }) => {
  const { nome, descricao } = item;
  return (
    <>
      <dt>{nome}</dt>
      <dd>{descricao}</dd>
    </>
  );
};

const Glossario = props => {
  const { items } = props;
  return (
    <dl>
      {items.map(item => (
        <ListItem item={item} key={item.id} />
      ))}
    </dl>
  );
};

export default Glossario;
```



Hands on!

# ACESSIBILIDADE

## FORMULÁRIOS

- Todos os elementos de um formulário HTML, como `<input>` e `<textarea>`, precisam ser **rotulados**. É necessário fornecer rótulos descritivos pois são expostos aos leitores de tela.
- Artigos que mostram como fornecer rótulos:
  - [The W3C shows us how to label elements](#)
  - [WebAIM shows us how to label elements](#)
  - [The Paciello Group explains accessible names](#)
- Em JSX é utilizado o atributo *htmlFor* para essa finalidade.

```
<label htmlFor="nomeDaEntrada">Nome:</label>  
<input id="nomeDaEntrada" type="text" name="nome"/>
```

# ACESSIBILIDADE

NOTIFICANDO ERROS AO USUÁRIO

- Situações de erro precisam ser entendidas por todos os usuários. Os artigos a seguir mostram como expor os erros aos leitores de tela:
  - [The W3C demonstrates user notifications](#)
  - [WebAIM looks at form validation](#)



# ACESSIBILIDADE

CONTROLE DE FOCO

- É importante que a aplicação web seja totalmente navegável apenas com o teclado.
- O artigo abaixo demonstra como isso deve ser feito.
  - [WebAIM talks about keyboard accessibility](#)

# ACESSIBILIDADE

FOCO NO TECLADO E FOCO DE CONTORNO

- Foco no teclado se refere ao elemento no DOM que foi selecionado e aceita ações do teclado.
- Somente use CSS que elimine este contorno, por exemplo, definindo `outline: 0`, se você for substituí-lo por outra implementação de esquema de foco.

```
import React from 'react';

const KeyboardFocus = () => {
  return <a href="https://portal.dataprev.gov.br">Teste de foco</a>;
};

export default KeyboardFocus;
```



Hands on!

# ACESSIBILIDADE

## MECANISMOS PARA PULAR CONTEÚDO

- São mecanismos para permitir que os usuários ignorem as seções de navegação anteriores em seu aplicativo, pois isso ajuda e acelera a navegação pelo teclado.
- **Links para pular navegação** são links ocultos que só se tornam visíveis quando os usuários interagem com a página usando o teclado. Eles são muito fáceis de implementar com alguns estilos e âncoras de páginas. Ver [esse](#) artigo.
- Também podem ser usados elementos e pontos de referência, como <main> e <aside>, para demarcar regiões de páginas como tecnologia assistiva, permitindo que o usuário navegue rapidamente para estas seções.

# ACESSIBILIDADE

MOVIMENTOS DO MOUSE E PONTEIRO (CURSOR)

- Todas as funcionalidades expostas através do movimento de mouse ou ponteiro também devem ser acessadas usando apenas o teclado.
- Os eventos onBlur e onFocus podem ser utilizados para substituir o click do mouse, quando necessário.
- Implementar os exemplos:
  - [Usando click](#)
  - [Usando onBlur e onFocus](#)



# ACESSIBILIDADE

## OUTROS PONTOS IMPORTANTES

- Definir corretamente o idioma (atributo ***lang***). Os leitores de tela levam em consideração essa informação para selecionar corretamente as configurações de voz.
- Definir o título do documento (tag <title>).
- Todo o texto legível na aplicação deve ter contraste de cores suficiente para permanecer legível ao máximo por usuários com baixa visão.
- Assistência ao desenvolvimento:
  - Plugin [eslint-plugin-jsx-a11y](#)

# DIVIDINDO O CÓDIGO

## EMPACOTAMENTO

- A maioria das aplicações React serão “empacotadas” usando ferramentas como Webpack, Rollup ou Browserify.
- Empacotamento (bundling) é o processo onde vários arquivos importados são unidos em um único arquivo: um “pacote” (bundle).
- Este pacote pode ser incluído em uma página web para carregar uma aplicação toda de uma vez.
- O Create React App, Next.js e Gatsby já disponibilizam uma configuração do Webpack pronta para empacotar a aplicação.

# DIVIDINDO O CÓDIGO

- Empacotamento é excelente, mas à medida que a aplicação cresce o pacote crescerá também. Especialmente se estiver usando grandes bibliotecas de terceiros.
- A divisão de código é um recurso suportado por empacotadores como Webpack, Rollup e Browserify (através de coeficiente de empacotamento (factor-bundle)) no qual pode-se criar múltiplos pacotes que podem ser carregados dinamicamente em tempo de execução.
- Dividir o código da aplicação pode ajudar a carregar somente o necessário ao usuário. Embora não se reduza a quantidade total de código da aplicação, se evita carregar código que o usuário talvez nunca precise.

# DIVIDINDO O CÓDIGO

import()

- A melhor forma de introduzir a divisão de código em uma aplicação é através da sintaxe dinâmica import().

```
import { add } from './math';  
console.log(add(16, 26));
```



```
import('./math').then(math => {  
  console.log(math.add(16, 26));  
});
```

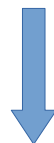


# DIVIDINDO O CÓDIGO

React.lazy

- A função do React.lazy é permitir renderizar uma importação dinâmica como se fosse um componente comum.

```
import OtherComponent from './OtherComponent';
```



```
const OtherComponent = React.lazy(() =>  
import('./OtherComponent'));
```

- Isto automaticamente carregará o pacote contendo o OtherComponent quando este componente for renderizado pela primeira vez.

# DIVIDINDO O CÓDIGO

React.lazy

- O componente lazy pode ser renderizado dentro de um componente Suspense, o que permite mostrar algum conteúdo de fallback (como um indicador de carregamento) enquanto aguardamos o carregamento do componente lazy.

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

const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <OtherComponent />
      </Suspense>
    </div>
  );
}
```

# DIVIDINDO O CÓDIGO

React.lazy

- A prop fallback aceita qualquer elemento React.
- O componente Suspense pode ser colocado em qualquer lugar acima do componente dinâmico.
- É possível ter vários componentes dinâmicos envolvidos em um único componente Suspense.

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

const OtherComponent = React.lazy(() => import('./OtherComponent'));
const AnotherComponent = React.lazy(() => import('./AnotherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <section>
          <OtherComponent />
          <AnotherComponent />
        </section>
      </Suspense>
    </div>
  );
}
```

# DIVIDINDO O CÓDIGO

DIVISÃO DE CÓDIGO BASEADA EM ROTAS

- Um bom lugar para começar é nas rotas.

```
import React, { Suspense, lazy } from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/About'));

const App = () => (
  <Router>
    <Suspense fallback={<div>Loading...</div>}>
      <Switch>
        <Route exact path="/" component={Home}/>
        <Route path="/about" component={About}/>
      </Switch>
    </Suspense>
  </Router>
);
```



# DIVIDINDO O CÓDIGO

## EXPORTAÇÕES NOMEADAS

- React.lazy atualmente suporta apenas export default.
- Se o módulo a ser importado usa exportações nomeadas, é possível criar um módulo intermediário que usa export default.

```
// ManyComponents.js  
export const MyComponent = /* ... */;  
export const MyUnusedComponent = /* ... */;
```

```
// MyComponent.js  
export { MyComponent as default } from "../ManyComponents.js";
```

```
// MyApp.js  
import React, { lazy } from 'react';  
const MyComponent = lazy(() => import("../MyComponent.js"));
```

# CONTEXTO

- Provê uma forma de passar dados entre a árvore de componentes sem precisar passar props manualmente em cada nível.
- Certos tipos de props (como preferências locais ou tema de UI), são utilizadas por muitos componentes dentro da aplicação.
- Contexto fornece a forma de compartilhar dados como esses, entre todos componentes da mesma árvore de componentes, sem precisar passar explicitamente props entre cada nível.

# CONTEXTO

QUANDO UTILIZAR

- É indicado para compartilhar dados que podem ser considerados “globais” para a árvore de componentes do React.
- Exemplos:
  - Usuário autenticado.
  - Tema da aplicação
  - Idioma selecionado

# CONTEXTO

EXEMPLO

```
import React from "react";

export const themes = {
  light: {
    foreground: "#000000",
    background: "#eeeeee",
  },
  dark: {
    foreground: "#ffffff",
    background: "#222222",
  },
};

// Contexto é criado com o tema 'dark' (default)
export const ThemeContext = React.createContext(themes.dark);
```



Hands on!



# CONTEXTO

EXEMPLO

```
import React from "react";

import { ThemeContext } from "../ThemeContext";

class ThemedButton extends React.Component {
  render() {
    let props = this.props;
    let theme = this.context;
    return (
      <button
        {...props}
        style={{ backgroundColor: theme.background }}
      />
    );
  }
}

ThemedButton.contextType = ThemeContext;

export default ThemedButton;
```



Hands on!

# CONTEXTO

## EXEMPLO

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

import ThemedButton from "../ThemedButton";
import { themes, ThemeContext } from "../ThemeContext";

const Toolbar = props => {
  return (
    <ThemedButton onClick={props.changeTheme}>Trocar Tema</ThemedButton>
  );
};

class Contexto extends Component {
  constructor(props) {
    super(props);
    this.state = {
      theme: themes.light,
    };
    this.toggleTheme = () => {
      this.setState(state => ({ theme: state.theme === themes.dark ? themes.light : themes.dark }));
    };
  }

  render() {
    return (
      <ThemeContext.Provider value={this.state.theme}>
        <Toolbar changeTheme={this.toggleTheme} />
        <ThemedButton>Botão</ThemedButton>
      </ThemeContext.Provider>
    );
  }
};

export default Contexto;
```



Hands on!

# ERROR BOUNDARIES

- Error boundaries são componentes React que capturam erros de JavaScript em qualquer lugar na sua árvore de componentes filhos, registram esses erros e mostram uma UI alternativa.
- Capturam estes erros durante a renderização, em métodos do ciclo de vida, e em construtores de toda a árvore abaixo delas.
- Não capturam erros em
  - Manipuladores de evento.
  - Código assíncrono (ex. callbacks de setTimeout ou requestAnimationFrame).
  - Renderização no servidor.
  - Erros lançados na própria error boundary (ao invés de em seus filhos).

# ERROR BOUNDARIES

- Um componente de classe se torna uma error boundary se ele definir um (ou ambos) dos métodos do ciclo de vida static `getDerivedStateFromError()` ou `componentDidCatch()`.
- O static `getDerivedStateFromError()` é usado para renderizar uma UI alternativa após o erro ter sido lançado.
- O `componentDidCatch()` é usado para registrar informações do erro.



# ERROR BOUNDARIES

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

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { error: null, errorInfo: null };
  }

  componentDidCatch(error, errorInfo) {
    this.setState({ error: error, errorInfo: errorInfo })
  }

  render() {
    if (this.state.errorInfo) {
      return (
        <div>
          <h2>Ocorreu um erro inesperado.</h2>
          <details style={{ whiteSpace: 'pre-wrap' }}>
            {this.state.error && this.state.error.toString()}
            <br />
            {this.state.errorInfo.componentStack}
          </details>
        </div>
      );
    }
    return this.props.children;
  }
}

export default ErrorBoundary;
```



Hands on!

# ERROR BOUNDARIES

```
import React from 'react';

import ErrorBoundary from './ErrorBoundary';
import BugSimulator from './BugSimulator';

const ErrorBoundaryView = () => {
  return (
    <ErrorBoundary>
      <BugSimulator />
    </ErrorBoundary>
  );
};

export default ErrorBoundaryView;
```

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

class BugSimulator extends Component {

  state = {
    handleError: false
  };

  handleSimulateError = () => {
    this.setState({
      handleError: true
    });
  }

  render() {
    if (this.state.handleError) {
      throw new Error("Erro simulado!");
    }
    return <button
      onClick={this.handleSimulateError}>Simula
      Erro</button>;
  }
};

export default BugSimulator;
```



Hands on!

# COMPONENTES DE ORDEM SUPERIOR (HOCs)

- Um componente de ordem superior (HOC, do inglês Higher-Order-Component) é uma técnica avançada do React para reutilizar a lógica de um componente.
- HOCs não são parte da API do React. Eles são um padrão que surgiu da própria natureza de composição do React.
- Concretamente, um componente de ordem superior é uma função que recebe um componente e retorna um novo componente.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

# COMPONENTES DE ORDEM SUPERIOR (HOCs)

## EXEMPLO

- O HOC a seguir exibe um componente de progresso enquanto os dados são carregados.
- Assim que os dados são carregados, o componente envolvido pelo HOC é exibido recebendo os dados via prop “data”.

```
import React from 'react';
import { CircularProgress } from "@material-ui/core";

export const withLoading = (WrappedComponent, isLoading, data, ...otherProps) => {
  return () => isLoading ?
    <CircularProgress /> :
    <WrappedComponent isLoading={isLoading} data={data} {...otherProps} />;
};
```





# COMPONENTES DE ORDEM SUPERIOR (HOCs)

## EXEMPLO

```
const UsersTable = props => {  
  const { data } = props;  
  return (  
    <TableContainer component={Paper}>  
      <Table aria-label="data Table" size="small">  
        <TableHead>  
          <TableRow>  
            <TableCell>ID</TableCell>  
            <TableCell>Nome</TableCell>  
            <TableCell>E-mail</TableCell>  
          </TableRow>  
        </TableHead>  
        <TableBody>  
          {data && data.length > 0 && data.map((row) => (  
            <TableRow key={row.id}>  
              <TableCell component="th" scope="row">{row.id}</TableCell>  
              <TableCell>{row.nome}</TableCell>  
              <TableCell>{row.email}</TableCell>  
            </TableRow>  
          ))}  
        </TableBody>  
      </Table>  
      {(data || data.length === 0) && (  
        <div style={styles.emptyMessage}>Nenhum usuário encontrado!</div>  
      )}  
    </TableContainer>  
  );  
};  
  
export default UsersTable;
```



Hands on!

# COMPONENTES DE ORDEM SUPERIOR (HOCs)

## EXEMPLO

```
import React from "react";
import { Grid } from "@material-ui/core";

const SystemLogs = props => {
  const { data } = props;

  const renderLogEntry = (entry, index) => {
    const { data, hora, mensagem } = entry;
    return (
      <Grid key={index} container>
        <Grid item sm={1}>{data}</Grid>
        <Grid item sm={1}>{hora}</Grid>
        <Grid item sm={10}>{mensagem}</Grid>
      </Grid>
    );
  };

  return (
    <div>
      <h2>Logs do Sistema</h2>
      {data && data.map(renderLogEntry)}
    </div>
  );
};

export default SystemLogs;
```



Hands on!

# COMPONENTES DE ORDEM SUPERIOR (HOCs)

## EXEMPLO

```
import React, { Component } from "react";
import UsersTable from "./UsersTable";
import { withLoading } from "./withLoading";

import jsonUsers from "./users.json";
import jsonLogs from "./logs.json";
import SystemLogs from "./SystemLogs";

class Hocs extends Component {
  state = {
    users: [],
    logs: []
  };

  fetchUsers = () => this.setState({ users: jsonUsers });
  fetchLogs = () => this.setState({ logs: jsonLogs });

  componentDidMount() {
    setTimeout(this.fetchUsers, 3000);
    setTimeout(this.fetchLogs, 5000);
  }

  render() {
    const { users, logs } = this.state;
    const isLoadingUsers = (!users || users.length === 0);
    const isLoadingSystemLogs = (!logs || logs.length === 0);
    const UsersTableWithLoading = withLoading(UsersTable, isLoadingUsers, users);
    const SystemLogsWithLoading = withLoading(SystemLogs, isLoadingSystemLogs, logs);
    return (
      <div>
        <UsersTableWithLoading />
        <br />
        <SystemLogsWithLoading />
      </div>
    );
  }
};

export default Hocs;
```



Hands on!

# PORTALS

- Portals fornece uma forma elegante de renderizar um elemento filho dentro de um nó DOM que existe fora da hierarquia do componente pai.

```
ReactDOM.createPortal(child, container)
```

- O primeiro argumento (child) é qualquer elemento React renderizável, como um elemento, string ou fragmento. O segundo argumento (container) é um elemento DOM.



# PORTALS

## UTILIZAÇÃO

- Normalmente, quando retornamos um elemento pelo método render de um componente ele é montado dentro do DOM como um filho do nó pai mais próximo.
- Entretanto, em algumas situações é útil inserir um elemento filho em um local diferente no DOM.

```
render() {  
  // React *não* cria uma nova div. Ele renderiza os filhos dentro do `domNode`.  
  // `domNode` é qualquer nó DOM válido, independente da sua localização no DOM.  
  return ReactDOM.createPortal(  
    this.props.children,  
    domNode  
  );  
}
```

# PORTALS

## UTILIZAÇÃO

- Um caso típico do uso de portals é quando um componente pai tem o estilo overflow: hidden ou z-index, mas você precisa que o filho visualmente “saia” desse container.
- Por exemplo, caixas de diálogo, hovercards, modals e tooltips.

# PORTALS

## EXEMPLO

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

import "./styles.scss";
import Modal from "./Modal";

class Portals extends Component {
  constructor(props) {
    super(props);
    this.state = { showModal: false };
    this.handleShow = this.handleShow.bind(this);
    this.handleHide = this.handleHide.bind(this);
  }

  handleShow() {
    this.setState({ showModal: true });
  }

  handleHide() {
    this.setState({ showModal: false });
  }

  render() {
    const modal = this.state.showModal ? (
      <Modal>
        <div className="modal">
          <div>Conteúdo renderizado fora da div pai</div>
          <button onClick={this.handleHide}>Fechar Modal</button>
        </div>
      </Modal>
    ) : null;

    return (
      <div className="app" id="app-root">
        <p>Esta div possui overflow: hidden.</p>
        <button onClick={this.handleShow}>Abrir Modal</button>
        {modal}
      </div>
    );
  }
}

export default Portals;
```



Hands on!

# PORTALS

## EXEMPLO

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

class Modal extends React.Component {
  constructor(props) {
    super(props);
    this.el = document.createElement("div");
  }

  getModalRootEl = () => {
    return document.getElementById("modal-root");
  }

  componentDidMount() {
    this.getModalRootEl().appendChild(this.el);
  }

  componentWillUnmount() {
    this.getModalRootEl().removeChild(this.el);
  }

  render() {
    return ReactDOM.createPortal(
      this.props.children,
      this.el,
    );
  }
}

export default Modal;
```

```
.app {
  height: 10em;
  width: 10em;
  background: lightblue;
  overflow: hidden;
}

.modal-root {
  position: relative;
  z-index: 999;
}

.modal {
  background-color: rgba(0, 0, 0, 0.5);
  position: fixed;
  height: 100%;
  width: 100%;
  top: 0;
  left: 0;
  display: flex;
  align-items: center;
  justify-content: center;
}
```



Hands on!



# CHECAGEM DE TIPOS COM PROPTYPES

```
cd nome-da-app  
npm install --save prop-types
```

- A biblioteca prop-types dispõe de um mecanismo que possibilita checar se as props passadas aos componentes são válida de acordo com o esperado.
- Possui uma variedade de validadores que podem ser usados para certificar que os dados recebidos pelo componentes são válidos.
- Realiza a validação em tempo de execução.

# CHECAGEM DE TIPOS COM PROPTYPES

## EXEMPLO

```
import React from "react";

import infoUsuarios from "./usuarios.json";
import UserInfo from "./UserInfo";

const PropTypesChecking = () => {

  const renderUsuario = usuario => {
    return <UserInfo key={usuario.id} {...usuario} />
  }

  return (
    <div>
      <h1>Usuários</h1>
      {infoUsuarios && infoUsuarios.map(renderUsuario)}
    </div>
  );
};

export default PropTypesChecking;
```

```
[
  {
    "id": 1,
    "nome": "Ricardo Glodzinski",
    "cpf": 12345678909,
    "idade": 34,
    "telefones": [
      {
        "id": 4,
        "ddd": 48,
        "numero": 38774990,
        "tipo": "PROFISSIONAL"
      },
      {
        "id": 5,
        "ddd": 48,
        "numero": 991887744,
        "tipo": "CELULAR"
      }
    ]
  },
  {
    "id": 2,
    "nome": "José da Silva",
    "cpf": 29694855517,
    "idade": 56,
    "telefones": [
      {
        "id": 1,
        "ddd": 21,
        "numero": 40953025,
        "tipo": "PROFISSIONAL"
      },
      {
        "id": 2,
        "ddd": 21,
        "numero": 987556622,
        "tipo": "CELULAR"
      }
    ]
  },
  {
    "id": 3,
    "ddd": 21,
    "numero": 30142558,
    "tipo": "RESIDENCIAL"
  }
]
]
```



Hands on!

# CHECAGEM DE TIPOS COM PROPTYPES

## EXEMPLO

```
import React from "react";
import PropTypes from "prop-types";

const fieldsetStyles = {
  border: "solid 1px #d1d2d3",
  padding: "10px",
  marginBottom: "15px"
};

const UserInfo = props => {
  const { id, nome, cpf, idade, telefones } = props;

  const renderTelefone = telefone => {
    const { id, ddd, numero } = telefone;
    return <div key={id}>({ddd}) {numero}</div>;
  };

  return (
    <fieldset style={fieldsetStyles}>
      <label>Informações do Usuários - ID {id}</label>
      <dl>
        <dt>Nome</dt><dd>{nome}</dd>
        <dt>CPF</dt><dd>{cpf}</dd>
        <dt>Idade</dt><dd>{idade}</dd>
      </dl>
      {telefones && telefones.length > 0 &&
        <fieldset style={fieldsetStyles}>
          <label>Telefones</label>
          {telefones.map(renderTelefone)}
        </fieldset>
      }
    </fieldset>
  );
};
```

```
UserInfo.propTypes = {
  id: PropTypes.number.isRequired,
  nome: PropTypes.string.isRequired,
  cpf: PropTypes.number.isRequired,
  idade: PropTypes.number.isRequired,
  telefones: PropTypes.arrayOf(PropTypes.shape({
    id: PropTypes.number.isRequired,
    ddd: PropTypes.number.isRequired,
    numero: PropTypes.number.isRequired,
    tipo: PropTypes.oneOf(["RESIDENCIAL",
      "PROFISSIONAL", "CELULAR", "RECADO"])
  })))
};

export default UserInfo;
```



Hands on!

# CHECAGEM DE TIPOS COM PROPTYPES

- Uma lista completa das validações disponíveis podem ser consultadas [aqui](#).
- É possível definir valores padrão (default) para as props através da atribuição à propriedade especial defaultProps.

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

```
Greeting.defaultProps = {  
  name: "Stranger"  
};
```





# Obrigado!

**Ricardo Glodzinski**

**Analista de Tecnologia da Informação**

ricardo.glodzinski@dataprev.gov.br

**Tiago de Andrade Freire**

**Analista de Tecnologia da Informação**

tiago.freire@dataprev.gov.br

Agosto de 2020



[www.facebook.com/dataprevtecnologia](https://www.facebook.com/dataprevtecnologia)



[@dataprev](https://twitter.com/dataprev)



[DATAPREV](https://www.linkedin.com/company/dataprev)