TRYBE
Modulo II – Front-end

Bloco 13 – Mais React

1) Melhorando o reuso de componentes: props.children e PropTypes

Ajuda no processo de projetos em comum com outros desenvolvedores.

Props.children

A/ Uso

Permite **flexibilizar a lógica de um componente**, garantindo que ele pode ser usado de várias formas e em vários contextos diferentes, sempre de forma legível e com sintaxe simples.

```
class Grid extends React.Component {
  render() {
    return <div>{this.props.children}</div>
  }
}
```

//componente pai renderizando filho

```
import React from 'react'

function GreetComponent(props) {
    return <div>
        <h1>Hello {props.name}</h1>
        {props.children}
    </div>
}

function App() {
    return (
        <GreetComponent name="Mehul">
            <p>This code still works!</p>
        </GreetComponent>
    )
}

export default App
```

Como usar props.children: *a diferença com outra props é que entende diretamente o que tem dentro do componente renderizado como seu child. Nem precisa escrever children="" na tag do componente.*

B/ Elementos filhos

→ Em React, elementos filhos não precisam ser componentes:

```
<Grid>
  Here is a row:
  <Row />
  Here is another row:
  <Row />
</Grid>
```

//pode misturar tipos de elementos filhos

→ Elemento filho pode ser uma function:

```
class Executioner extends React.Component {
  render() {
    // Percebe que estamos executando uma função aqui?
    //                          ↓
    return this.props.children()
  }
}
```

// a usar depois deste jeito

→

```
<Executioner>
  {() => <h1>Hello World!</h1>}
</Executioner>
```

Útil por exemplo para fazer requisição de API com fetch dentro de um componente.

→ Diferentes tipos de filhos em React: **'children is opaque'**

Podem ser qualquer tipo de objeto JavaScript, assim como arrays, funções ou objeto.

React.Children permite manipular todos sem if.

Para iterar, **React.Children.map** e **React.Children.forEach**.

```
<div>
  {React.Children.only(this.props.children)}
</div>
```

*//**React.Children.only** para garantir que tenha apenas um child*

```
Total Children: { Children.count(this.props.children) }
```

// para contar os children qualquer seja o tipo deles (lengthquebraria)

```
React.Children.toArray(children)
```

//Returns the children opaque data structure as a flat array with keys assigned to each child.

Ver todas functions React.Children.something aqui .

PropTypes, checagem de tipos

Usamos para **verificar se as props estão sendo passadas corretamente.**
A checagem de tipos com PropTypes **garante que não ocorram erros de lógica facilmente evitáveis**, relacionados aos tipos das props passadas para um componente.
//Em algumas aplicações, você pode usar extensões do JavaScript como Flow ou TypeScript para checar os tipos de toda a sua aplicação.//

Instalar
Comando *npm install --save prop-types*
Não esquecer de importar no arquivo trabalhado: *import PropTypes from 'prop-types';*

Checkagens possíveis

```
import PropTypes from 'prop-types';

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

Greeting.propTypes = {
  name: PropTypes.string
};
```

→ ***TIPO*** de uma prop no uso de um componente;

```
ComponentName.propTypes = {
  propname: PropTypes.string
};
```

Outros tipos:
.array .bool .func .number .object .symbol

→ garantir a ***presença*** *de props obrigatórias* no uso de um componente;
Adicionar *.isRequired*

→ checar que uma prop é um ***objeto*** de formato específico;

```
// An array of a certain type
  optionalArrayOf: PropTypes.arrayOf(PropTypes.number),
```

→ garantir que uma prop é um ***array*** com elementos de um determinado tipo.

```
  // An object with property values of a certain type
  optionalObjectOf: PropTypes.objectOf(PropTypes.number),
```

→ dar valor **Padrão**

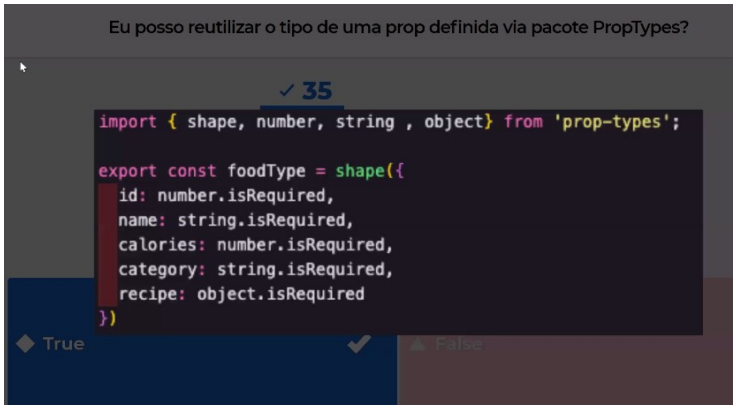
```
ComponentName.defaultProps = {
  propname: 'defaultvalueyouchoose'
};
```

→ exigir **apenas um elemento filho** (Component incluindo um this.props.children)

```
ComponentName.propTypes = {
  children: PropTypes.element
};
```

→ Todas checkagens listadas aqui e aqui.

Como funciona: ao passar uma props inválida ou incorreta, aparece um erro no console mas a aplicação não quebra.
Estrutura:

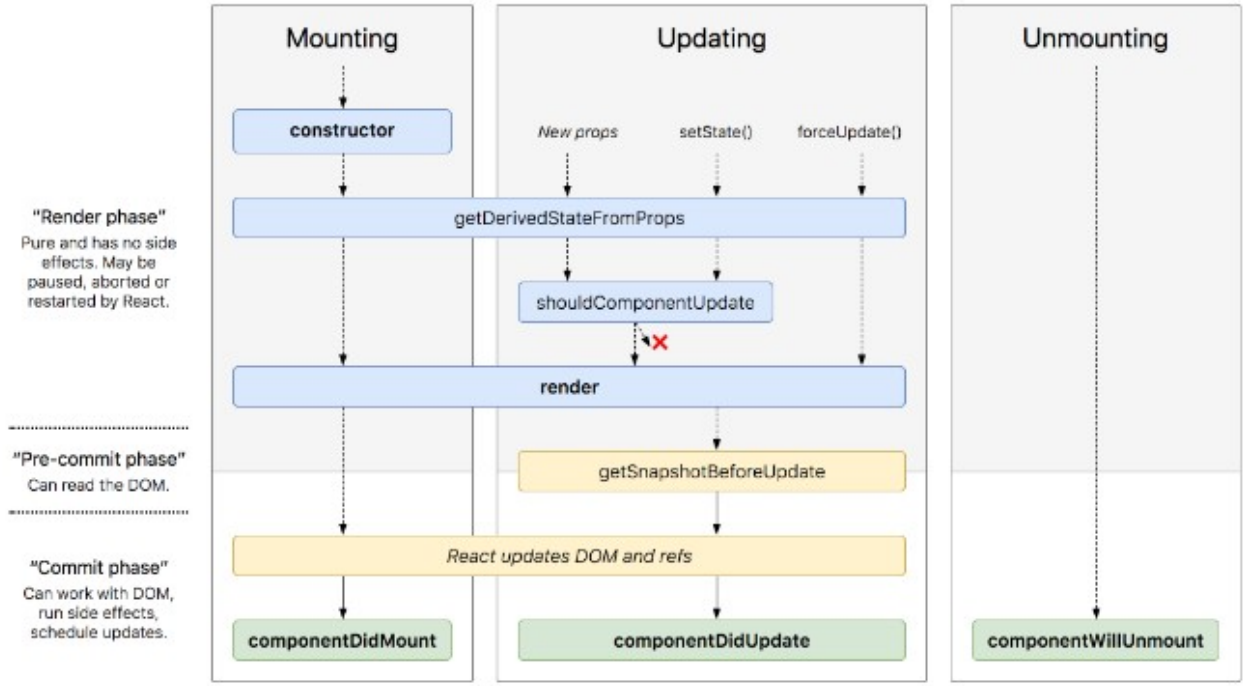


---

2) Ciclo de vida de componentes em React

As funções de ciclo de vida do componente vêm para nos **dar o controle necessário para utilizar cada recurso no momento certo** e para garantir que a assincronicidade do React não prejudique a lógica do que você está tentando executar.

4 etapas: iniciar, mounting, update, unmounting.
Cada uma **acessível por meio de métodos nativos** dos class components.



1. Inicialização

Quando o **componente recebe as props e os estados**.
Método:
constructor - recebe as props e define o estado.

2. Montagem (mounting)

Quando o componente é **inserido no DOM**.
Métodos:
render - renderiza o componente, inserindo-o no DOM;

componentDidMount - dispara uma ou mais ações após o componente ser inserido no DOM

→ bom para API, setInterval contador, addEventListener… e pode chamar setState apenas dentro de condicional para evitar loop infinito.

```
componentDidMount() {
  fetch("https://api.example.com/items")
    .then(res => res.json())
    .then(
      (result) => {
        this.setState({
          isLoaded: true,
          items: result.items
        });
      },
      // Nota: É importante lidar com os erros aqui
      // em vez de um bloco catch() para não recebermos
      // exceções de erros dos componentes.
      (error) => {
        this.setState({
          isLoaded: true,
          error
        });
      }
    )
}
```

3. Atualização (update)

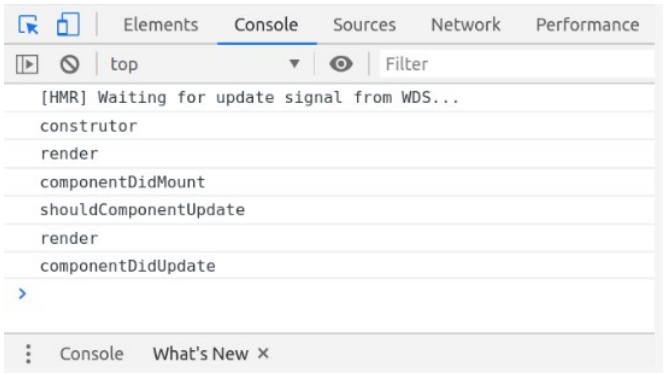
Quando os **props ou estados do componente são alterados.**
Métodos:
- *shouldComponentUpdate(nextProps, nextState)* - possibilita autorizar ou não o componente a atualizar (permite comparar os atuais e próximos estados ou props e adicionar a lógica, tem valor default true, retorna boolean);

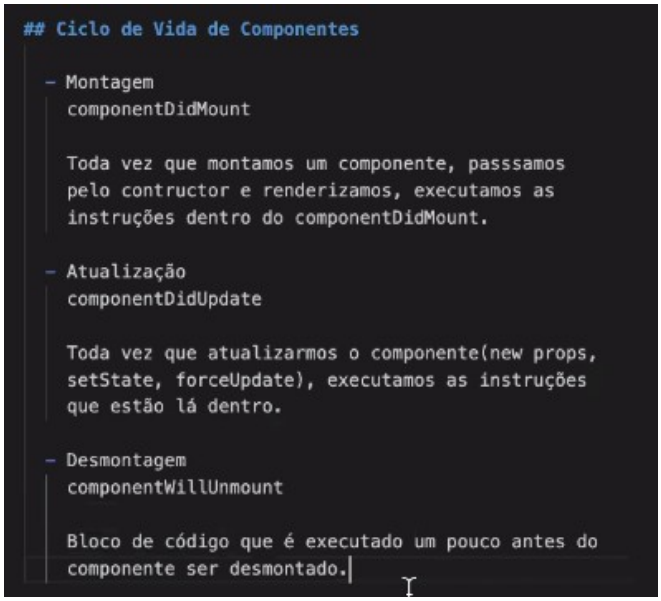
- *componentDidUpdate(prevProps, prevState)* - dispara uma ou mais ações após o componente ser atualizado;

4. Desmontagem (unmounting)

Quando o componente **morre, sumindo do DOM**.
Método:
componentWillUmount - dispara uma ou mais ações antes de o componente ser desmontado.

Ordem de execução





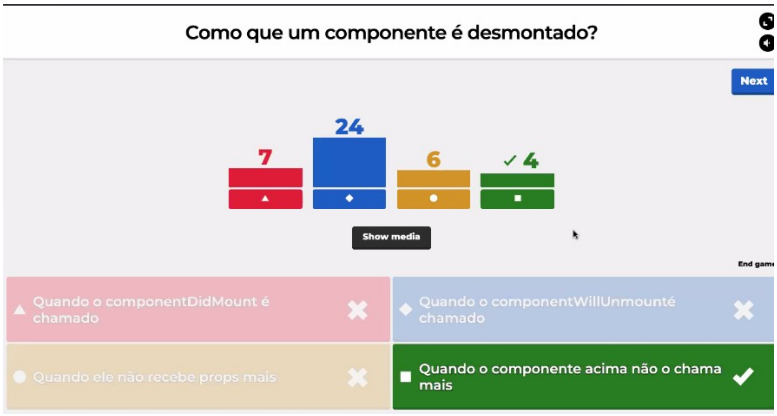
Métodos raramente usados:

getDerivedStateFromProps(props, state): oportunidade de transferir valor de props para state. Se não quiser renderizar nenhum estado novo, simplesmente return null.

getSnapshotBeforeUpdate(prevProps, prevState)

ComponentDidCatch(error, info)

Dicas diversas



---

3) React Router

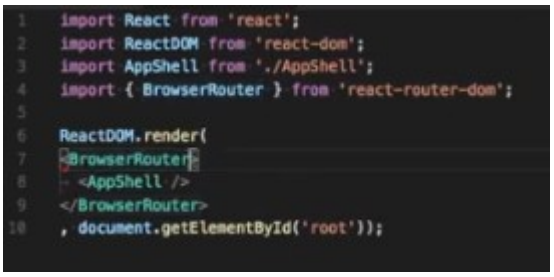
Biblioteca de componentes React que permite tornar a aplicação React **navegável**, entregando assim uma **melhor experiência para o user**.

Instalar
npm install react-router-dom
ou *npm install --save-dev react-router-dom* (informa que tem que baixar essa dependência junto)

Single Page Application

Aplicações em React Router são SPAs.
Os recursos do React carregam de uma vez. Troca de dados com cliente bem rápida. Suave.

Componentes BrowserRouter e Route



BrowserRouter encapsula a aplicação, de forma a possibilitar fazer uso de navegação:
- *import { BrowserRouter } from 'react-router-dom';*
- tag do App renderizado entre tag do BrowserRouter.

Route é para criar rotas, mapeando o caminho da URL com o componente correspondente.

- Sintax trabalhando dentro do App, para escrever o começo do caminho:
<Route path="caminhoQueQuer" />
Acessar vendo *localhost:3000/caminhoQueQuer*

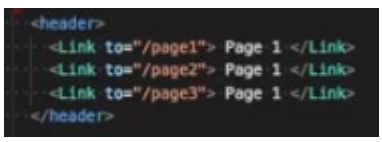
- Sintax trabalhando dentro do App, para caso de correspondência exata com o caminho:
*<Route **exact** path="caminho" />*

- Sintax trabalhando com props children:
```
<Route path="/about" >
    <About />
</Route>
```

- Para dar rota **default**: *path="/"*

Componente Link

Para **criar links de navegação** na aplicação, sem recarregar toda a página. Usar no lugar da tag a:



<Link to="/caminhoPertinente" > Legenda que quiser inserir </Link>

Componentes Route com passagem de props

Pode associar um componente com o caminho da URL **via children, component ou render**.

Como escolher
É necessário passar informações adicionais via props para o componente a ser mapeado?
→ NÃO
Usar props **component**
component={Movies}
→ SIM
Usar props **render**
<Route path="/movies" render={(props) => <Movies {...props} movies={['Cars', 'Toy Story', 'The Hobbit']} />} />

Ambos têm informações de rota (*match, location e history*).

```
function Content() {
  const sampleData = ['12.1', '12.2', '12.3'];
  return (
    <main>
      <Switch>
        <Route exact path='/' component={Home} />
        <Route path='/solutions'
               render={props => <Solutions {...props} avaiableSolutions=
               {sampleData} />} />
        <Route path='/trybe-content' component={TrybeContent} />
        <Route path='/schedule' component={Schedule} />
        <Route path='/setup' component={Setup} />
      </Switch>
    </main>
  );
}
```

Componentes Route com passagem de parâmetro (rotas dinâmicas)

Vantagem de podermos utilizar o **mesmo componente** para renderizar **vários caminhos** diferentes.

Sintax:
:nome_do_parametro dentro da propriedade path

<Route path="/movies/:movieId" component={Movie} />
// aqui def param movieId e Movie renderizado em qualquer caminho iniciando com /movies
// Dica: sempre especificar primeiro

this.props.match.params.*movieId* para pegar a parte dinâmica da URL.

Componente Switch

Para **encapsular, estruturar e organizar as rotas** da aplicação.

```
<main>
  <Switch>
    <Route path="/page1" component={View1}/>
    <Route path="/page2" component={View1}/>
    <Route path="/page3" component={View1}/>
  </Switch>
</main>
```

Procura rotas de cima para baixo, por isso, recomenda-se definir as **rotas mais específicas primeiro** e deixar as mais genéricas por último.

```jsx
<main>
  <Switch>
    <Route exact path='/' component={Home} />
    <Route path="/solutions/:solutionId" component={Solutions}/>
    <Route path='/solutions'
           render={props => <Solutions {...props} avaiableSolutions={sampleData} />} />
    <Route path='/trybe-content' component={TrybeContent} />
    <Route path='/schedule' component={Schedule} />
    <Route path='/setup' component={Setup} />

    {/* Exemplo com props.children do Staroscky e do Lucas abaixo, acessem com localhost:3000/HELLO */}
    <Route to="/HELLO" >
      OI
    </Route>

  </Switch>
</main>
```

// switch ordenado

Componente Redirect

Para alternar entre rotas: permite **ativamente levar quem usa** a aplicação para diferentes locais dela.

<Redirect to="/somewhere/else" />

Todas as sintax aqui, dependendo do tipo de objeto ou local onde quer redirigir.

Dicas diversas

React CHEATSHEET http://www.developer-cheatsheets.com/react
–

!Ordenar bem o Switch para que um elemento não englobe o outro e acabe ignorando uma route.