TRYBE
Modulo II – Front-end

Bloco 12 – React com eventos

Criar interação no React. “Components defined as classes have some additional features. Local state is a feature available only to class Components.”

1) Componentes com estado

Um estado representa um momento de um componente dinâmico. State é um **objeto**.

Na classe: constructor e this.state

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
 render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

this.state inicializa o estado no construtor.

Rendendo: ciclo de vida do componente, this.setState

Montar e desmontar
Montar é configurar a atualização, desmontar é limpar.
Método *componentDidMount()* é executado depois que a saída do componente é renderizada no DOM.
Derrubado via *componentWillUnmount()*.

this.setState() é para agendar atualizações para o estado local do componente. *setState()* agenda uma atualização para o objeto state de um componente.
Quando o state muda, o componente responde renderizando novamente.
Aceita objeto como paramêtro.

```
class Clock extends React.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>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Dicas de uso e entendimento do State

a) Não modificar o state diretamente

```
// Errado
this.state.comment = 'Hello';
```

```
// Correto
this.setState({comment: 'Hello'});
```

this.state pode ser atribuido apenas no construtor.

b) SetState pode ser assincrono

Ex: dentro de manipuladores de evento.
Para consertar, usar function (aqui, sintax arrow), para **encadeiar atualizações**.

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

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

c) Os dois estados mergeam

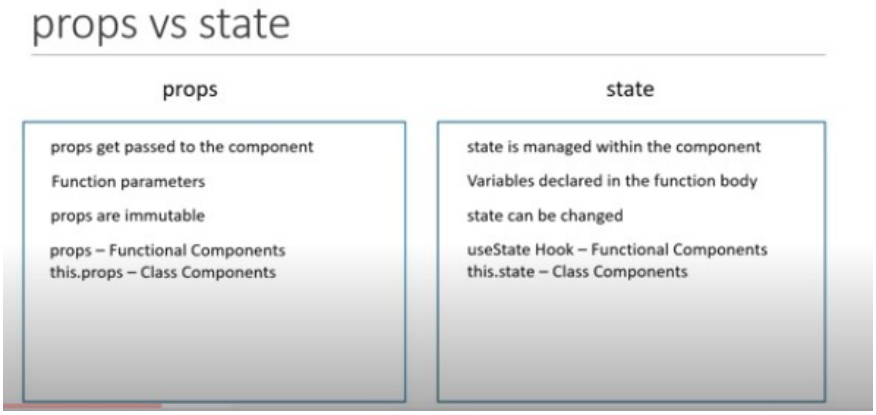
Chamando setState(), o React **mescla o objeto que você fornece ao state atual**.

d) Dados fluem para baixo

State é local / encapsulado: não é accessível para componentes que nem definem nem possuim. Afeta apenas componentes abaixo da árvore.

Dicas diversas

arrow function escrito sem ‘const’ inicial no React
–
Vantagem de react e estado: o que é atualizado é apenas aquele elemento, o que faz com que os carregamentos e aplicativos são mais rápidos
–
Diferenças entre props e state



---

2) Eventos e formulários no React

Manipular eventos

- Eventos no React são os mesmos que no html com sintax **camelCase**: exemplo onClick.

- **e.preventDefault()** para não ter que se preocupar mais com diferenças de navegadores e não redirecionar a cada interação (e sendo evento em parâmetro da function).

- Ficar de olho no **bind do this** para que o evento funcione:

Quando dá undefined, dois caminhos possíveis:

1. Sintax com bind explicito no construtor:
this.functionname = this.functionname.bind(this);
Para quando chamamos a function em questão, ela saiba onde está.

2. Escrever function com sintax arrow
Quando declara com function, o escopo da função é o que está dentro da function, já o escopo de arrow é o de onde está inserido.
E lembrar de integrar no evento também
<button onClick={() => this.handleClick()}>

- **Passar argumentos** para manipuladores de eventos

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

- **Montar e desmontar** eventlistener

```
componentDidMount() {
document.addEventListener("keydown", this.handleKeyPress);
  }
  componentWillUnmount() {
document.removeEventListener("keydown", this.handleKeyPress);
  }
```

Forms

A. Controlled components

Quando o input do user vira parte do state, então React controla o valor deste campo de inputs. O state React state vira assim “a única fonte de verdade”.

```
this.state = {value: ''};
this.setState({value: event.target.value});
<input type="text" value={this.state.value} onChange={this.handleChange} />
```

Um input de elemento de form controlado deste jeito por React é chamado de “controlled component”.

B. Tags em componentes controlados

<input type="text">, <textarea>, <select> funcionam de um jeito similar: todas aceitam um atributo **value** onde pode **implementar um componente** controlado.

```
<textarea value={this.state.value}
onChange={this.handleChange} />
```

```
<select value={this.state.value}
onChange={this.handleChange}>
    <option> </option>
</select>
```

```
class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isGoing: true,
      numberOfGuests: 2
    };

    this.handleInputChange = this.handleInputChange.bind(this);
  }

  handleInputChange(event) {
    const target = event.target;
    const value = target.name === 'isGoing' ? target.checked : target.value;
    const name = target.name;

    this.setState({
      [name]: value
    });
  }

  render() {
    return (
      <form>
        <label>
          Is going:
          <input
            name="isGoing"
            type="checkbox"
            checked={this.state.isGoing}
            onChange={this.handleInputChange} />
        </label>
        <br />
        <label>
          Number of guests:
          <input
            name="numberOfGuests"
            type="number"
            value={this.state.numberOfGuests}
            onChange={this.handleInputChange} />
        </label>
      </form>
    );
  }
}
```

Atributo *name*: para **gerir diversos inputs de uma vez** via event.target.name.

const { name, value } = event.target;

```
changeHandler = event => {
  console.log(event.target);
  const { name, value } = event.target;
  this.setState({
    [name]: value
  });
}
```

//function changeHandler atendendo todos inputs, toda vez que atualiza

C. Uncontrolled components

Basicamente bom saber que existem, apesar da maioria das vezes utilizarmos controlled.

"In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself."

- sintax de **ref** para pegar os valores do form no DOM. `this.input = React.createRef();`

- **default values** para especificar o valor inicial sem controlar os updates.

```
<input defaultValue="Bob" type="text"  ref={this.input} />
```

→ Quando usar controlled ou uncontrolled?

O artigo-fonte também avisa que é fácil passar de uncontrolled para controlled se precisar.

feature	uncontrolled	controlled
one-time value retrieval (e.g. on submit)	✅	✅
validating on submit	✅	✅
instant field validation	❌	✅
conditionally disabling submit button	❌	✅
enforcing input format	❌	✅
several inputs for one piece of data	❌	✅
dynamic inputs	❌	✅

Dicas diversas

Componente pronto para **validar o form** →

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

class FormErrors extends Component {
  constructor(props) {
    super(props)
  }

  render() {
    // formErrors: {email: '', password: ''}
    const { formErrors } = this.props;
    return(
        <div>
          {Object.keys(formErrors).map((fieldName, i) => {
            if(formErrors[fieldName].length === 0) return ''
            return <p key={i}>{fieldName} {formErrors[fieldName]}</p>
          })}
        </div>
    )
  }
}

export default FormErrors;
```

+ colocar no construtor

```
formErrors: {email: '', password: ''}
```

+ integrar na function

```
changeHandler = event => {
  console.log(event.target);
  const { name, value } = event.target;
  this.setState({
    [name]: value,
    formErrors: {
      ...state.formErrors,
      [name]: this.validateField(name, value)
    }
  });
}
```

+ renderizar no app

Exemplo de form do FCC

```
class MyForm extends React.Component {
constructor(props) {
super(props);
this.state = {
input: '',
submit: ''
};
this.handleChange = this.handleChange.bind(this);
this.handleSubmit = this.handleSubmit.bind(this);
}
handleChange(event) {
this.setState({
input: event.target.value
});
}
handleSubmit(event) {
event.preventDefault()
this.setState({
submit: this.state.input
});
}
render() {
return (
<div>
<form onSubmit={this.handleSubmit}>
<input
value={this.state.input}
onChange={this.handleChange} />
<button type='submit'>Submit!</button>
</form>
<h1>{this.state.submit}</h1>
</div>
);
}
};
```