descomplica

React – Formulários

Prof: Ayrton Teshima

Nessa aula você aprendeu

Formulários é um tópico à parte pois possui suas peculiaridades. Até agora vimos como definir estado em um componente, mas quando falamos de formulários e os inputs que inserimos informações, eles possuem o seu próprio estado, o que torna "difícil" de controlar no React.

Quando trabalhamos com inputs, muitas vezes queremos:

- Aplicar máscara enquanto o usuário digita no campo (ex: máscara de telefone e cpf);
- Aplicar validações informando se o dado é válido;
- Fazer requisições para serviços enquanto é digitado (ex: autocomplete).

E é claro, queremos ter todas as informações acessíveis para o envio do formulário. Tendo tudo isso em mente, como conseguimos obter as informações digitadas caracter por caracter, em um campo de texto para aplicar uma máscara ou validação?

É aí que entram os componentes controlados e não controlados.

Componentes controlados

Componentes controlados é a principal forma de utilizar formulários no React. Aqui você de fato controla cada campo de um formulário, ignorando o seu estado próprio. Você controla o valor que vai ser digitado em cada input. Dessa forma, conseguimos ter o controle e aplicar as máscaras, validações e etc.

Vamos começar criando um componente básico de formulário, com uma tag *form* e um *input text*:

```
export const Formulario = (props) => {
  return (
    <form>
      <input type="text" name="nome" />
    </form>
  )
}
```

Veja, temos um *input text* aqui, mas não temos acesso ao conteúdo que foi inserido nele. Você pode digitar à vontade, mas, como fazemos para capturar o valor digitado?

Precisamos deixar ele controlado pelo React. Para conseguir isso, precisamos controlar cada valor digitado no campo. A forma de fazê-lo, é passar duas propriedades para cada input e utilizar o estado do componente para gerenciar.

A primeira ação que vamos fazer é utilizar o *useState* para:

1. Gerar o estado com o valor renderizado no campo;
2. Gerar método que vai modificar o valor renderizado.

Utilizando o *state* vamos ter controle total do valor do campo

```
import { useState } from 'react';

export const Formulario = (props) => {
  const [nomeValor, setNomeValor] = useState(''); // definimos como uma
string vazia
  return (
    <form>
      <input type="text" name="nome" />
    </form>
  )
}
```

Depois, precisamos fazer o *input text* renderizar o valor da *const nomeValor*. Para tal, basta adicionarmos a propriedade *value* do *input,* passando a constante *nomeValor.*

```
import { useState } from 'react';

export const Formulario = (props) => {
  const [nomeValor, setNomeValor] = useState('');
  return (
    <form>
      <input type="text" name="nome" value={nomeValor} />
    </form>
  )
}
```

Faça o seguinte teste: tente digitar qualquer valor no campo de nome, você vai ver que o valor não é renderizado, pois ele está fixado com o valor da constante *nomeValor*, que é um texto vazio.

O que precisamos fazer agora? Simples, fazer com que, toda vez que o usuário tente digitar um valor no campo, ele pegue e atualize o estado *nomeValor*. Como fazemos isso? Utilizando o evento *onChange* do *input*, este evento é acionado toda vez que ocorre uma alteração no campo. Toda vez que esse evento acontecer, a gente pega o valor digitado e invoca a função *setNomeValor* para atualizar o estado *nomeValor*.

```
import { useState } from 'react';

export const Formulario = (props) => {
  const [nomeValor, setNomeValor] = useState('');

  // Função que vai ser invocada toda vez que o usuário digitar no input
  const handleChange = (event) => {
    setNomeValor(e.target.value);
  };

  return (
    <form>
      <input
        type="text"
        name="nome"
        value={nomeValor}
        onChange={handleChange} // Evento que é invocado toda vez que o
campo altera (ou deveria alterar)
       />
    </form>
  )
}
```

Quando invocamos a função de estado *setNomeValor*, nós pegamos o valor que está atualmente no campo de nome e atualizamos o estado, com isso o estado é atualizado e a constante *nomeValor* é renderizado no campo.

Ou seja, para o valor digitado ser realmente renderizado no campo, ele antes é atualizado no estado do componente para depois ser refletido no campo, com isso temos o controle total do input.

Múltiplos campos

Talvez você já esteja se perguntando: E se meu formulário tiver muitos campos, preciso criar um estado para cada? A resposta é não.

Você consegue criar uma estrutura em um único estado, com isso utilizamos um único hook *useState*. Vamos criar mais um campo no formulário para simular isso:

```
<form>
  <input
    type="text"
    name="nome"
    value={nomeValor}
    onChange={handleChange}
   />
  <input type="text" name="cidade" /> {/* <-- Campo cidade criado */}
</form>
```

Campo cidade criado, mas ainda não passamos as propriedades *value* e *onChange*, pois antes de fazer, vamos refatorar nosso estado para agora receber um objeto e não o valor do campo nome.

Objetos no JavaScript servem para guardar estrutura de dados diversificados. Vamos utilizar essa estrutura para armazenar todos os valores que nosso formulário precisar. Então, vamos alterar o valor que passamos para o *useState,* definindo um objeto com os valores de cada campo como um texto vazio.

```
const [campoValor, setCampoValor] = useState({
  nome: '',
  cidade: ''
});
```

O próximo passo é fazer com que nossos campos recebem o seu valor através da nova constante de estado *campoValor*.

```
<form>
  <input
    type="text"
    name="nome"
    value={campoValor.nome}
    onChange={handleChange}
   />
  <input type="text" name="cidade" value={campoValor.cidade} />
</form>
```

Temos nossos campos lendo a informação do estado a partir de um objeto. Agora falta refatorar nossa função *handleChange*.

Toda vez que executamos *setCampoValor*, passamos um objeto (lembre-se que agora o seu valor deve ser um objeto). O objetivo é atualizar o novo valor do campo, para conseguir isso, precisamos saber qual o campo que queremos atualizar o seu valor.

Pegar o valor nós já sabemos como, é através do *event.target.value*, quando acessamos *event.target*, temos acesso ao elemento input que estamos manipulando.

Observe que utilizamos no objeto do estado as propriedades com o mesmo nome dos input (propriedade *name* do input). Já que temos acesso ao elemento via JavaScript, podemos acessar suas propriedades simplesmente adicionando o ponto (.) e o nome da propriedade, que no caso é *name*.

```
const handleChange = (event) => {
  setCampoValor({
    ...campoValor,
    [event.target.name]: event.target.value
  })
};
```

Para atualizar o objeto do state, passamos um objeto como argumento do *setCampoValor* e mergeamos o valor do estado atual dentro dele. Em seguida, passamos dinamicamente o valor do nome da propriedade como chave do objeto, por isso colocamos entre colchetes. Depois, demos dois pontos e definimos seu valor.

Agora sim, falta passar o evento *onChange* com a função *handleChange* no campo de cidade, podemos ver o código todo agora:

```
import { useState } from 'react';

export const Formulario = (props) => {
  const [campoValor, setCampoValor] = useState({
    nome: '',
    cidade: ''
  });

  const handleChange = (event) => {
    setCampoValor({
      ...campoValor,
      [event.target.name]: event.target.value
    })
  };

  return (
    <form>
      <input
        type="text"
        name="nome"
        value={campoValor.nome}
        onChange={handleChange}
       />
      <input
        type="text"
        name="cidade"
        value={campoValor.cidade}
        onChange={handleChange}
       />
    </form>
  )
}
```

Para fecharmos, geralmente queremos obter o valor todo de todos os inputs na hora que vamos enviar o formulário. Para fazer isso, simplesmente adicione o evento *onSubmit* na tag *form* e passe a função que vai ser invocada toda vez que o formulário for enviado.

```
export const Formulario = (props) => {
  // ... funções anteriores aqui

  // Função que vai ser invocada quando formulário for enviado
  const handleSubmit = (event) => {
    event.preventDefault();

    console.log(campoValor);
  }

  return (
    <form onSubmit={handleSubmit}>
      {/* campos aqui */}
    </form>
  )
}
```

A função *handleSubmit,* que é invocada, recebe o objeto *event* (assim como toda função ouvinte de evento). Assim que inicializamos a função, invocamos a função *preventDefault* do *event* para suspender o comportamento padrão de um formulário, que é fazer o seu envio real (não queremos que o formulário realmente seja enviado, queremos apenas capturar o evento de submit dele).

Depois disso, acessamos o objeto do estado *campoValor*. Nesse momento, podemos fazer o que quiser com os dados, seja para enviar para uma API externa ou salvar em um storage do navegador.

Componentes não controlados

Como podemos ver nos componentes controlados, a maior parte das vezes você vai criar formulários controlados, pois é nesse formato que conseguimos capturar caracter por caracter digitado e realizar algumas ações, como validações, máscara, entre outros.

Mas, caso precise, existe a opção de utilizar componentes não controlados em seus formulários. A consequência direta disso é que você só vai conseguir obter os dados mediante ao envio do formulário (ou algum outro evento que você especifique).

Vamos voltar ao código anterior e refatorar para componentes não controlados

```
export const Formulario = (props) => {
  const handleSubmit = (event) => {
    event.preventDefault();

    console.log(campoValor);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="nome"
       />
      <input
        type="text"
        name="cidade"
       />
    </form>
  )
}
```

Veja que fizemos uma grande limpeza no nosso código. Como não vamos mais manipular as informações no estado, apagamos o estado gerado pelo *useState* e, também, a função *handleChange*, assim como as propriedades *value* e *onChange* dos *input*.

O nosso único desafio aqui é conseguir capturar os valores presentes em cada campo no momento do submit do formulário. Existem algumas formas de fazer isso, mas vamos focar na forma React de fazer.

Sempre que temos um elemento React, ele é apenas um elemento React. Vão ter vezes que vamos querer ter acesso real ao elemento (DOM API) que renderiza no navegador e, o React nos proporciona esse acesso através dos *refs*.

Afinal, para que precisamos ter acesso real aos elementos, neste caso, aos campos? Precisamos, pois, no evento de *submit* do formulário, acessaremos esses campos e pegaremos seus valores.

Para cada elemento que definimos no React, podemos passar uma propriedade para ele chamada *ref* e seu valor tem que ser um *ref* específico que o React conhece. Esse valor nós obtemos através de um hook chamado *useRef*

```
import { useRef } from 'react'; // importamos o hook

export const Formulario = (props) => {
  // Definimos um ref para cada campo, seu valor inicial é null pois aqui
estamos referenciando um elemento DOM e não o valor do input
  const nomeRef = useRef(null);
  const cidadeRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();

  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="nome"
        ref={nomeRef} // definimos que esse input é o nomeRef
       />
      <input
        type="text"
        name="cidade"
        ref={cidadeRef} // definimos que esse input é o cidadeRef
       />
    </form>
  )
}
```

Com os *refs* definidos e sendo passado para seus elementos, podemos agora acessar sempre que quisermos os elementos reais (DOM), através dos refs *nomeRef* e *cidadeRef*.

```
import { useRef } from 'react';

export const Formulario = (props) => {
  const nomeRef = useRef(null);
  const cidadeRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();

    // os ref recebem um objeto e quando associado a um elemento, ele
preenche a propriedade current com o elemento
    console.log(nomeRef.current.value, cidadeRef.current.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="nome"
        ref={nomeRef}
       />
      <input
        type="text"
        name="cidade"
        ref={cidadeRef}
       />
    </form>
  )
}
```