Como fazer um CRUD com create-react-app

Sequência para criar um sistema de cadastro

Dan Vitoriano Oct 16, 2017 · 6 min read

Configurando ambiente

Para usar o create-react-app você precisa ter o Node e o NPM instalado na sua máquina.

JSX e Babel

JSX é um JavaScript extendido, que suporta tags de XML. É mais fácil do que escrever JavaScript ao natural.

É preciso usar um transpilador como o Babel para ler a sintaxe mais nova e converter para a padrão dos navegadores.

Plugins do Babel como React Extension Converter e ES2015 Converter, interpretam o JSX e o converte para JavaScript entendível pelo navegador.

Por exemplo, *import* e *class* são sintaxes do ES2015 que precisam ser transpiladas.

Webpack

É um *module bundle*, como o *Browserify*.

Permite gerar um *bundle*, compilando seus *assets* dentro de um único arquivo. Por exemplo, converte CSS, SVG em um JS, e insere ele dentro do seu arquivo HTML.

Usando CSS

Pure.css

Podemos usar o Pure.css, um framework compacto com grids, buttons e outros elementos prontos.

Quem importa o CSS dentro do React é o Webpack.

Toda tag em JSX precisa ser fechada:

```
<img src={imageSrc} alt={imageName} />
```

A palavra `class` é reservada no ES6, então você deve usar `className` :

```
<img src={imageSrc} className={nomeClasse} />
```

Comentários

```
{/* comments */}
```

Importando Componentes

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

Ao importar o `React` é um módulo default de `react` , e o `Component` dentro das chaves é algo a mais que você deseja importar.

Classes

Você pode criar uma classe para definir seu componente, e ela ser uma extensão do `Component` puro do React.

Sem JSX

```
const Menu = React.createClass({
  render(){
    return()
  }
});
export default Menu;
```

Com JSX

```
class Menu extends Component{
  render(){
    return()
  }
}
export default Menu;
```

Render

É na função `render(){}` que tudo acontece. Pode ser escrita também como `render: function(){}` .

Dentro do `Render` tudo que deve aparecer vai dentro do `return()` .

ReactDOM Render

Com o ReactDOM.render você renderiza os componentes importados criados em classes exportadas:

Sem JSX

```
import Menu from './Menu';

ReactDOM.render(
  React.createElement(Menu)
);
```

Com JSX

```
import Menu from './Menu';

ReactDOM.render(
  <Menu/>
);
```

Orientação a Objetos no React

> *Comportamento + Estado*

Estado dos Componentes

Você só pode guardar estado no React na variável que ele te traz chamada `state`. Você usa um construtor do ES6, para definir o estado inicial do componente.

Construtor

No seu construtor você deve inicializar o estado das coisas. Para usar o **`this`** no seu construtor, primeiro você precisa chamar **`super()`**. Depois você atribui seu JSON ao estado desse `this`. No nosso caso, o JSON inicial será um objeto vazio:

```
constructor(){
  super();
  this.state = {};
}
```

Atribuindo uma lista ao estado inicial

Se você quiser atribuir uma lista ao estado inicial do seu componente, dentro da lista, crie seu objeto:

```
this.state = {
  lista : [ {nome: 'danilo', email: 'dan@dan.com'} ]
}
```

Utilizando código dinâmico

Para renderizar seu código dinâmico dentro do seu componente, utiliza as chaves `{ ... }` .

```
<tbody>
  { this.state.lista }
</tbody>
```

Para mapear o array da sua lista, utilize a função **map** do ES6:

```
<tbody>
  {
    this.state.lista.map(function(obj){
      return (
        <tr>
          <td>{obj.nome}</td>
          <td>{obj.email}</td>
        </tr>
      );
    })
  }
</tbody>
```

Utilizando jQuery para fazer requisições XHR

```
npm install jquery --save
```

No código:

```
import $ from 'jquery';
```

Atualização do estado dos componentes

Algumas funções do React facilitam identificar quando o estado deve ser alterado.

componentDidMount()

Logo após o `render` ser executado, significa que o *componente acabou de ser montado*, esta função será executada. Você pode usá-la logo após o construtor:

```
componentDidMount(){
  $.ajax({
    ... sua função ajax
  });
}
```

componentWillMount()

Logo antes do `render` ser executado, significa que o *componente ainda será montado*, e esta função será executada antes.

> *O ideal é que toda vez que o* `state` *mudar, o* `render()` *execute novamente.*

setState

Toda vez que você quiser alterar o estado de um componente, utilize `setState` . Porém, ao usá-lo dentro do jQuery, você precisa fazer um `bind()` informando que o `this` é referente ao React, e não o `this` do jQuery:

```
componentDidMount(){
  $.ajax({
    url:"http://cdc-react.herokuapp.com/api/autores",
    dataType: 'json',
    success: function(response){
      this.setState({lista:reponse});
    }.bind(this)
  });
}
```

No exemplo acima, a cada vez que você fizer uma requisição, e ela retornar uma resposta com sucesso, "defina o novo estado" (`setState`) com a resposta da sua requisição ajax.

Ciclo de Vida do DOM

Adicionar propriedades `key` aos seus elementos ajuda o React a entender quais partes ele deve renderizar toda vez que você atualizar o estado:

```
<tbody>
  {
    this.state.lista.map(function(obj){
      return (
        <tr key={obj.id}>
          <td>{obj.nome}</td>
          <td>{obj.email}</td>
        </tr>
      );
    })
  }
</tbody>
```

> *Você não altera seu componente. Você apenas atualiza o estado.*

Eventos

O React traz eventos próprios que serão mapeados depois de compilados para eventos do DOM, são os **`SyntheticEvents`** - eventos do React que mapeiam para eventos reais:

onSubmit

```
<form onSubmit={this.enviaForm} method="post">...</form>
```

onChange

```
<Input id="id" type="email" name="email" value={this.state.email}
onChange={this.setEmail} label="email" />
```

Função — Enviar dados de um formulário disparado por um evento

Um exemplo de como declarar a função disparada pelo evento para enviar um formulário, usando XHR Ajax com jQuery, prevenindo bubbling e usando **JSON.stringify** para enviar os dados através de um formulário:

```
enviaForm(evento){
    evento.preventDefault();
    $.ajax({
      url:'yourapi.com',
      contentType:'application/json',
      dataType:'json',
      type:'post',
      data: JSON.stringify({nome:''}),
      success: function(resposta){
        console.log(resposta);
      },
      error: function(resposta){
        console.log(resposta);
      }
    });
  }
```

No construtor, lembre-se de fazer o bind do `this` :

```
this.enviaForm = this.enviaForm.bind(this)
```

Manutenção do estado

> *Não manipulamos elementos DOM diretamente. Modificamos o `state`. O responsável por manipular o elemento DOM é o React.*

Portanto, para enviarmos dados de um formulário devemos enviar o estado do campo que foi preenchido:

```
data: JSON.stringify({
  nome:this.state.nome
})
```

No construtor, adicione o campo ao `state` :

```
this.state = {lista: [], nome:''}
```

Estado dos campos de um formulário

É preciso definir que o valor de um campo está relacionado ao seu estado (**`{this.state.nome}`**) e que ao ser alterado deve disparar um evento que muda seu estado (**`onChange={this.setNome}`**):

```
<input id="nome" type="text" name="nome" value={this.state.nome}
onChange={this.setNome} />
```

A função de `set` seria assim:

```
setNome(evento){
  this.setState({nome:evento.target.value});
}
```

É preciso atribuir o bind ao `this` no construtor:

```
this.setNome = this.setNome.bind(this)
```

Atualizando uma lista dinamicamente ao enviar um formulário sem recarregar a página

No nosso CRUD, para atualizar uma lista com os dados enviados pelo formulário, devemos apenas atribuir a resposta do envio da requisição Ajax ao estado da lista:

```
success: function(resposta){
   this.setState({lista:resposta});
```

```
  }.bind(this)
```

Não esqueça de fazer o *bind* do `this` para o jQuery.

Reutilizando componentes

Uma das grandes vantagens do React é a possibilidade de componentização, permitindo assim sua reutilização.

Propriedades

Ao criarmos um componente para ser reutilizado, passaremos parâmetros que serão recebidos no componente por meio de um atributo herdado da classe `Component` chamado `props`.

```
import React, { Component } from `react`;
class CustomInput extends Component{

  render() {
    return(
      <label htmlFor={this.props.id}>{this.props.label}</label>
      <input id={this.props.id} type={this.props.type} name=
{this.props.nome} value={this.props.value} onChange=
{this.props.onChange}/>
    );
  }
}
```

O seu componente agora pode ser importado em outro componente e, passando as devidas propriedades, reutilizado:

```
<CustomInput id="nome" type="text" name="nome" value=
{this.state.nome} onChange={this.setNome} label="Nome"/>
```

Repare que o atributo `label` faz parte do componente, mas no DOM, será renderizado como um elemento `<label>`.

High-order Components

> *São os componentes responsáveis por encapsular um estado que será trabalhado por vários outros componentes e que comumente nomeamos utilizando o sufixo* `Box`.

Os argumentos que você passa pra dentro de um componente, ficam disponíveis sempre dentro de uma variável `props` . Seu componente deve definir de onde ele buscará seu estado no High-order component:

```
export default class AutorBox extends Component {
...
  <FormularioAutor/>
  <TabelaAutores lista={this.state.lista} />
...
}
```

E utilizar as propriedades no componente de origem para que elas consigam passar este estado. As `props` tem um JSON que é criado dinamicamente em função dos argumentos que são passados:

```
{
  this.props.lista.map(function(autor){
    return(
      ...
    );
  }
}
```

No componente FormularioAutor do seu High-order component, criamos um argumento apenas para dizer que ele "precisa atualizar a listagem":

```
<FormularioAutor callbackAtualizaListagem=
{this.atualizaListagem}/>
```

E assim definimos a função `atualizaListagem` , sem esquecer de dar bind no `this` no construtor:

```
constructor() {
  ...
  this.atualizaListagem = this.atualizaListagem.bind(this);
}

atualizaListagem(novaLista) {
  this.setState({lista:novaLista});
}
```

No componente original FormularioAutor, o retorno da requisição Ajax usará props para atualizar a `callbackAtualizaListagem`:

```
success: function(resposta){
  this.props.callbackAtualizaListagem(resposta);
}.bind(this)
```

React Create React App