04

Evoluindo o componente de cadastro de usuário

Transcrição

Nesta aula, evoluiremos nosso componente de cadastro do usuário e implementaremos o comportamento esperado a ele. Precisaremos definir uma forma do usuário acessar a página de cadastro, incluir seus dados no formulário e então finalizar o registro na aplicação.

Começaremos pelo formulário, e neste momento não deveremos nos preocupar com o nome das classes CSS que serão utilizadas nem com a estrutura HTML, afinal todas essas informações são provenientes do Bootstrap, e isso não é nosso foco no momento.

Em `NovoUsuario.vue`, inseriremos uma `<div>` no lugar do `<h1>Novo usuário.</h1>`. Esta terá uma `class` igual a `"form-group"`. Dentro, teremos uma `<label>` que vai ser no nosso `Nome`, o teremos o `<input>` do tipo `"text"` com a classe `"form-control"`.

```
<template>
    <div class="form.group">
        <label for="nome">Nome</label>
        <input type="text" class="form-control">
    </div>
</template>
```

COPIAR CÓDIGO

Vamos salvar e ver como ficou no navegador.

Já conseguimos ver o *input* com o "Nome", mas está ocupando toda a extensão da tela, o que arrumaremos depois.

Além do nome, precisaremos do email e da senha. Vamos copiar e colar o bloco anterior duas vezes, e mudaremos o `<label>` para `"email"` com `E-mail` , e a senha `"senha"` , e o `<input>` será do tipo `"password"` .

```
<template>
    <div class="form.group">
        <label for="nome">Nome</label>
        <input type="text" class="form-control">
    </div>
    <div class="form.group">
        <label for="email">E-mail</label>
        <input type="email" class="form-control">
    </div>
    <div class="form.group">
        <label for="senha">Senha</label>
        <input type="password" class="form-control">
    </div>
</template>
```

COPIAR CÓDIGO

Ao executarmos, receberemos um erro do Vue, nos dizendo que o template raiz precisa ter apenas um elemento importado, e se olharmos no `<template>` , notaremos que estamos importando três `<div>` .

Ao criarmos um componente, é importante que encapsulemos todo nosso HTML e exportar apenas um elemento, caso o contrário teremos erros no código.

Recortaremos todo o conteúdo de `<template>` com "Ctrl + X" e criaremos uma nova `<div>` com `<form>` , retirando o `action` porque não o usaremos agora.

Em seguida, colaremos o bloco recortado dentro de `<form>` e adicionaremos uma nova classe `"container"` . No `<h1>` da linha seguinte, escreveremos `Novo Usuário` e vamos salvar.

```
<template>
    <div class="container">
        <h1>Novo Usuario</h1>
        <form>
            <div class="form.group">
                <label for="nome">Nome</label>
                <input type="text" class="form-control">
            </div>
            <div class="form.group">
                <label for="email">E-mail</label>
                <input type="email" class="form-control">
            </div>
            <div class="form.group">
                <label for="senha">Senha</label>
                <input type="password" class="form-control">
            </div>
        </form>
    </div>
</template>
```

COPIAR CÓDIGO

De volta ao navegador, veremos os campos de "Nome", "E-mail" e de "Senha" conforme o esperado.

Em seguida, voltaremos ao código e colocaremos um `<button>` de `Salvar` cuja `class` será `"btn btn-primary"` e o tipo será `"submit"` .

```
<template>
    <div class="container">
        <h1>Novo Usuario</h1>
        <form>
```

```
            <div class="form.group">
                <label for="nome">Nome</label>
                <input type="text" class="form-control">
            </div>
            <div class="form.group">
                <label for="email">E-mail</label>
                <input type="email" class="form-control">
            </div>
            <div class="form.group">
                <label for="senha">Senha</label>
                <input type="password" class="form-control">
            </div>
            <button class="btn btn-primary"
type="submit">Salvar</button>
        </form>
    </div>
</template>
```

COPIAR CÓDIGO

Salvaremos e, no navegador, veremos toda a estrutura do formulário funcionando e exportando somente um elemento raiz `"container"` e o botão de "Salvar".

Agora que já definimos o HTML do nosso componente, precisamos dar vida à ele. Faremos uma conexão desses inputs com os nossos dados do usuário.

Dentro de um arquivo `.vue`, teremos a tag `<template>` com todo o visual do nosso componente, e então criaremos também a tag `<script>` em que exportaremos o objeto que terá todas as configurações do componente, seguindo o padrão indicado pelo Vue.

A primeira configuração que faremos diz respeito aos dados. Queremos que sempre que o componente for renderizado os dados únicos não se misturem com o restante da aplicação.

Criaremos a propriedade `data:` , a qual é uma `function ()` que retorna um objeto, onde definiremos as propriedades que queremos. Primeiro, o `usuario:` será um objeto com `nome:` , `senha:` e `email:` .

```
//código anterior omitido

<script>
export default {

  data: function() {
    return {
      usuario: {
        nome: "",
        senha: "",
        email: ""
      }
    }
  },
</script>
```

COPIAR CÓDIGO

Agora precisaremos conectar nossos inputs com as propriedades deste objeto usando `v-model` , indicando que o modelo deste input é `"usuario.nome"` , do input seguinte é `"usuario.email"` e o terceiro é `"usuario.senha"` .

Uma vez que a ligação foi realizada, realizaremos a submissão do formulário. O Vue.js nos ajuda neste caso, e falaremos que queremos tratar o `@submit` de `<form>` e, além disso, escreveremos `.prevent` para evitarmos que o formulário tenha o comportamento padrão do html 5, que é enviar os dados e recarregar a página.

Assim, irá chamar e tratar a função `enviarformulario` que definiremos. Com isso, indicaremos que a execute toda vez que um usuário clicar no botão de submeter formulário.

```
<template>
  <div class="container">
    <h1>Novo usuário</h1>
    <form @submit.prevent="enviarFormulario">
      <div class="form-group">
        <label for="nome">Nome</label>
        <input type="text" class="form-control" v-
model="usuario.nome" />
      </div>
        <div class="form.group">
            <label for="senha">Senha</label>
            <input type="password" class="form-control" v-
model="usuario.senha">
        </div>
        <button class="btn btn-primary"
type="submit">Salvar</button>
     </form>
  </div>
</template>

<script>
export default {

  data: function() {
    return {
      usuario: {
        nome: "",
        senha: "",
        email: ""
      }
    }
  },
</script>
```

COPIAR CÓDIGO

Assim como implementamos nosso objeto de dados, temos os `methods:` que também são objetos. Definiremos a função `enviarFormulario:` com a `function`

`()`.

Isso funciona, porém é muito comum usarmos a forma nova de definir uma propriedade de um objeto que é uma função nos componentes que temos em produção. É mais curta e fica desta forma: `enviarFormulario ()`.

Faremos o mesmo efeito definindo uma função, assim como fizemos com nosso objeto de dados. Na nova sintaxe do ECMA, não precisamos mais explicitar que se trata de uma função.

Quando o usuário submeter o formulário, faremos um `console.log()` do nosso `usuario` usando `this`.

```
//código anterior omitido

<script>
export default {

  data: function() {
    return {
      usuario: {
        nome: "",
        senha: "",
        email: ""
      }
    }
  },
  methods: {
    enviarFormulario () {
        console.log(this.usuario)
    }
}
</script>
```

COPIAR CÓDIGO

Ao preenchermos o formulário que criamos no navegador e fizermos o *submit*, veremos no Console do inspetor de código que os dados são enviados para o local esperado, e então temos acesso às propriedades `nome:` , `email:` e `senha:` .

Com o formulário funcional e capturando o evento de envio, podemos começar a enviar os dados para o `back-end` .