06

Trabalhando com recursos

Transcrição

Aprendemos a utilizar $http para consumir API's. No entanto, $http é um objeto especializado apenas na realização de requisições assíncronas ou se você preferir, requisições Ajax. Contudo, há um outro objeto mais especializado focado no consumo de API's que seguem o Padrão REST. É o **$resource**.

Quando trabalhamos com Single Page Applications, é extremanente comum acessarmos API's criadas com o padrão REST. REST é um estilo arquitetural para construir aplicações em rede que utiliza nada mais nada menos que o próprio protocolo http para sua comunicação. Resumidamente, cada recurso que desejamos acessar possui um identificador único e usamos os métodos (GET, PUT, POST, entre outros) do http para executarmos operações.

Geralmente é a equipe do backend que constroi as API's disponibilizando os endereços para a equipe front-end poder consumir. Como este é um curso focado em front, a construção de uma API REST esta fora do escopo. Na Alura você encontra cursos de REST, inclusive com linguagens diferentes, caso queria se aprofundar no assunto.

Então, como a API disponibilizada pelo curso segue o padrão REST, podemos usar `$resouce` ao invés de `$http` .

Primeiro, precisamos ter um recurso configurado. Vamos alterar o componente `Home` e no método `created` vamos adicionar dinamicamente a propriedade `resource` em nosso componente. Ela armazenará o retorno de

`this.$resource` que nada mais é do que um recurso configurado para um endereço:

```
// alurapic/src/components/home/Home.vue

// código anterior omitido
  created() {


    // agora conseguimos acessar o recurso configurado em
outros métodos do nosso componente
    this.resource = this.$resource('v1/fotos');

    this.$http
      .get('v1/fotos')
      .then(res => res.json())
      .then(fotos => this.fotos = fotos, err =>
console.log(err));

  }
}
// código posterior omitido
```

COPIAR CÓDIGO

Veja que imediatamente abaixo da nova propriedade, há o código que usa `$http` para buscar as fotos da nossa API. Vamos usar `this.resource` em seu lugar. Por convenção, se chamarmos o método `query` do nosso recurso, estaremos fazendo um requisição do tipo `GET` para `v1/fotos` :

```
// alurapic/src/components/home/Home.vue

// código anterior omitido
  created() {
```

```
      this.resource = this.$resource('v1/fotos');

      this.resource
        .query()
        .then(res => res.json())
        .then(fotos => this.fotos = fotos, err =>
console.log(err));

      /*
      this.$http
        .get('v1/fotos')
        .then(res => res.json())
        .then(fotos => this.fotos = fotos, err =>
console.log(err));
        */
    }
}
// código posterior omitido
```

COPIAR CÓDIGO

Não parece ter mudado muita coisa, mas para quem chama o método `query` não precisa saber o detalhe do endereço, apenas no momento da criação do recurso. Isso se tornará mais evidente agora, quando formos alterar o código que apaga um foto da nossa API.

Sabemos que endereços como `v1/fotos/3` quando usado o método DELETE apaga um recurso da noss API. Claro, ela esta preparada para entender endereços como esse inclusive o método empregado.

Eu acabei de dizer para vocês que o endereço da nosso recurso será definido em um lugar apenas, em `this.resource` . Mas e agora? Precisamos de um endereço que aceite receber o código da foto para que seja apagada.

Para isso, precisamos alterar o endereço passado para `$resource` adicionando, um curinga, ou melhor, um parâmetro:

fotos`:

```
// alurapic/src/components/home/Home.vue

// código anterior omitido
  created() {

    // veja que foi adicionado {/id}
    this.resource = this.$resource('v1/fotos{/id}');

    this.resource
      .query()
      .then(res => res.json())
      .then(fotos => this.fotos = fotos, err =>
console.log(err));
  }
}
// código posterior omitido
```

COPIAR CÓDIGO

Esse parâmetro do endereço da nossa API, quando não for passado, é ignorado. Perfeito para o método `query` que só leva em consideração `v1/fotos`, excluindo o parâmetro.

Agora, para apagarmos um recurso com o `this.resource` que criamos, fazemos da seguinte forma:

```
// alurapic/src/components/home/Home.vue

// código anterior omitido

 methods: {

    remove(foto) {
```

```
      // a chave do objeto é o parâmetro usando no endereço
do recurso

      this.resource
        .delete({id: foto._id})
        .then(
          () => {
            let indice = this.fotos.indexOf(foto);
            this.fotos.splice(indice, 1);
            this.mensagem = 'Foto removida com sucesso'
          },
          err => {
            this.mensagem = 'Não foi possível remover a
foto';
            console.log(err);
          }
        )
    }

  },

  created() {

    // parametrizando o endereço

    this.resource = this.$resource('v1/fotos{/id}');

    this.resource
      .query()
      .then(res => res.json())
      .then(fotos => this.fotos = fotos, err =>
console.log(err));
  }
}
// código posterior omitido
```

COPIAR CÓDIGO

Veja que a função `this.resource.delete` recebe como primeiro parâmetro um objeto JavaScript. A chave do objeto tem que ser exatamente igual ao parâmetro que adicionamos em nossa rota, no caso `id`. Para esta chave, passamos o valor. O resource sabe adicionar o parâmetro passado para completar o endereço.

Por fim, vamos alterar o componente `Cadastro` e substiuir `$http` pelo `$resource`. Precisamos adicionar o método `created` para configurarmos nosso resource.

Por fim, usamos o método `save` para realizarmos um post para nossa API:

```
<!-- alurapic/src/components/cadastro/Cadastro.vue -->

<template>

 <!-- código omitido -->

</template>

<script>

import ImagemResponsiva from '../shared/imagem-
responsiva/ImagemResponsiva.vue'
import Botao from '../shared/botao/Botao.vue';
import Foto from '../../domain/foto/Foto.js';

export default {

  components: {

    'imagem-responsiva': ImagemResponsiva,
    'meu-botao': Botao
  },

  data() {
```

```
    return {

      foto: new Foto(),
      resource: {}
    }
  },

  methods: {

    grava() {

      console.log(this.foto);

      // o método save realiza um POST por debaixo dos panos
enviado os dados passado como parâmetro
      this.resource
        .save(this.foto)
        .then(() => this.foto = new Foto(), err =>
console.log(err));

    }
  },

  created() {

    this.resource = this.$resource('v1/fotos{/id}');
  }

}

</script>
<style scoped>

/* código omitido */

</style>
```

COPIAR CÓDIGO

Veja que $resource foi feito para consumir API's no padrão REST. Se a API das suas aplicações não usa esse padrão, você pode usar o `$http` sem problema algum.

Apesar do nosso código estar um pouquinho mais sofisticado ele ainda deixa a desejar. Veja que tanto em `Home` quanto em `Cadastro` definimos o recurso no método `created` . E se o endereço do recurso mudar? Precisaremos mudar em vários lugares.

O ideal é centralizar todo acesso para API's em uma classe com essa responsabilidade que possui diversas vantagens. É isso que veremos no próximo vídeo.