02

Foto ruim? Não rasgue, apague!

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/mean-js/stages/06-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo. Só não esqueça de baixar as dependências do projeto no terminal com o comando `npm install`.

Foto ruim? Não rasgue, apague!

Temos apenas uma lista com duas fotos fornecida pelo nosso servidor. Mas se enjoarmos de uma delas? Que tal implementarmos em nosso servidor a rota de exclusão? Aquela que será chamada pelo Angular quando clicarmos no botão `Remover` da página principal.

É claro, não queremos clicar em uma foto e remover todas as outras, por isso nosso servidor precisa receber o ID da foto que desejamos remover. Criar uma rota parametrizada com um curinga e obter o ID enviado **não** é novidade para nós, fizemos isso no capítulo anterior.

Ainda lembra em qual arquivo configuramos as rotas que dão acesso à nossa API de fotos? Se não lembra, "recordar é viver" e o nome do arquivo é `alurapic/app/routes/foto.js`. Vamos editá-lo:

```
// alurapic/app/routes/fotos.js

module.exports = function(app) {

    var api = app.api.foto;

    app.get('/v1/fotos', api.lista);

    app.get('/v1/fotos/:id', api.buscaPorId);
};
```

Ampliando nosso vocabulário com novos verbos

Bom, agora vem a dúvida. Para acessarmos uma foto dado seu id usamos a URL `/v1/fotos/:id`, isso significa que precisaremos usar outra URL como `v1/fotos/remover/1`, certo? Não necessariamente. Quando queremos obter uma foto usamos o ver GET, mas quando formos remover? Usaremos o mesmo verbo? Não, existe um específico para deleção de recursos, o verbo DELETE.

Aprendemos que o Express possui uma função para cada verbo do protocolo http, sendo assim temos:

```
module.exports = function(app) {

    var api = app.api.foto;

    app.get('/v1/fotos', api.lista);

    app.get('/v1/fotos/:id', api.buscaPorId);

        // lidando como verbo DELETE
```

```
    app.delete('/v1/fotos/:id', api.removePorId);
};
```

Filtrando o indesejado

Por mais que tenhamos duas URL's iguais, dependendo do verbo empregado o Express chamará uma ou outra rota. Perfeito, não? Agora, precisamos implementar `api.removePorId` em `alurapic/app/api/fotos.js` :

```
// alurapic/app/api/fotos.js

var fotos = [
    {_id: 1, titulo: 'Leão', url:'http://www.fundosanimais.com/Minis/leoes.jpg' },
    {_id: 2, titulo: 'Leão 2', url:'http://www.fundosanimais.com/Minis/leoes.jpg' }
];

var api = {};

api.lista = function(req, res) {

    res.json(fotos);
};

api.buscaPorId = function(req, res) {

    var foto = fotos.find(function(foto) {
        return foto._id == req.params.id;
    });

    res.json(foto);
};

api.removePorId = function(req, res) {

       // criando uma nova lista sem a foto com o ID procurado
    fotos = fotos.filter(function(foto) {
        return foto._id != req.params.id;
    });
};

module.exports = api;
```

E viva o Javascript! Eu filtrei a lista de fotos barrando aquela foto com o ID que desejamos remover da nova lista criada, tudo através da função `filter` .

Uma resposta sutil, em códigos

Será que estamos esquecendo de alguma coisa? Claro que sim! Apagamos a foto, mas ficou faltando enviarmos um resposta. Neste caso, não queremos enviar um JSON ou qualquer outro dado. E agora? Precisamos enviar algo, ou pelo menos sinalizar o status da operação em nosso servidor. O que enviar?

O protocolo http não define apenas verbos, mas também códigos que possuem significados únicos dentro da especificação. Por exemplo, recebemos 404 para um recurso não encontrado e 500 para um erro interno no servidor. Porém, um que cai

igual a uma luva é o código **204**. Este código indica que o servidor executou a operação, mas nenhum informação foi retornada. Enviando um "status code" através da função **res.sendStatus**:

```
// alurapic/app/api/fotos.js

// código anterior omitido

api.removePorId = function(req, res) {

    fotos = fotos.filter(function(foto) {
        return foto._id != req.params.id;
    });

       // enviando resposta, mas apenas código de status
    res.sendStatus(204);
};
```

Elegância até na definição de rotas

Reiniciando...abrindo URL `localhost:3000` e removendo a primera foto! Voalá, funcionou! Isso significa que já podemos avançar mais com nosso server? Quase, primeiro quero lhe mostrar algo.

Vamos abrir para edição nosso arquivo de rotas `alurapic/app/routes/foto.js` . Você consegue enxergar algo duplicado nele?

```
module.exports = function(app) {

    var api = app.api.foto;

    app.get('/v1/fotos', api.lista);

    app.get('/v1/fotos/:id', api.buscaPorId);

    app.delete('/v1/fotos/:id', api.removePorId);
};
```

Perceba que tanto na busca quanto na deleção temos a mesma URL, a diferença mora apenas no verbo empregado. Muito provavelmente, quando eu for alterar a URL que obtém fotos, eu também vou querer alterar a URL que remove uma foto. Você vai lembrar de alterar no outro lugar? Nem eu!

Para resolver problemas como este e simplificar nosso código, podemos codificar da seguinte forma:

```
module.exports = function(app) {

    var api = app.api.foto;

    app.get('/v1/fotos', api.lista);

    // apenas uma URL, dois verbos distintos
    app.route('/v1/fotos/:id')
        .get(api.buscaPorId)
        .delete(api.removePorId);
};
```

Usamos `app.route` para definirmos uma rota. Em seguida, encadeamos uma chamada à função `get` e outra à função `delete`. Cada uma delas recebeu sua respectiva função da API de fotos. Agora, quando mudarmos a URL, mudaremos em um lugar apenas.