03

Faça sua escolha e conheça o Curinga!

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/mean-js/stages/05-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` .

Nossa aplicação em AngularJS pelo menos já consegue exibir uma lista de fotos enviada pelo servidor. Aliás, esse envio não foi uma "geração expontânea". Tivemos que criar uma rota do Express responsável em fornecer essas fotos. Nosso próximo passo será criar um recurso da nossa API que seja capaz de buscar uma foto pelo seu ID. Esse passo é importante, porque é através dessa rota que nossa aplicação Angular será capaz de selecionar a foto que desejamos editar.

Saindo da encruzilhada

Por exemplo, temos duas fotos exibidas na página principal. A primeira tem ID 1 e a segunda ID 2. Dessa forma, quando clicarmos em cada uma das fotos, nossa aplicação Angular realizará as seguintes requisições:

```
localhost:3000/v1/fotos/1
localhost:3000/v1/fotos/2
```

Veja que essa rota é muito parecida com `v1/fotos` , com a diferença de possuir um número que representa o ID da foto. Inclusive, das duas requisições de exemplo, a única coisa que muda é o ID da foto. Uma solução é criarmos uma rota para cada um desses ID's. Vamos editar `alurapic/app/routes/fotos.js` e fazermos um teste. Como é um teste, vamos colocar a lógica de resposta diretamente, e não em sua API:

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

module.exports = function(app) {

    var api = app.api.foto;

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

    app.get('/v1/fotos/1', function(req, res) {
        console.log('busca foto com ID 1');
    });
    app.get('/v1/fotos/2', function(req, res) {
        console.log('busca foto com ID 2');
    });

};
```

Tudo bem que não damos uma resposta ainda, mas reiniciando e testando vemos que a mensagem do servidor é exibida corretamente para as URL's:

```
localhost:3000/v1/fotos/1
localhost:3000/v1/fotos/2
```

Curinga ou Coringa?

E agora? Se tivermos 1000 fotos? Teremos 1000 rotas? Com certeza não. Uma solução é pensar no ID da foto que faz parte da URL como um curinga, algo que pode variar. Felizmente o Express nos permite fazer isso. Vamos deixar uma rota apenas, indicando que o ID é um curinga:

```
// alurapic/app/routes/foto.js

module.exports = function(app) {

    var api = app.api.foto;

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

    app.get('/v1/fotos/:id', function(req, res) {
        console.log('busca foto com ID 1');
    });
};
```

Usamos `:id` no lugar do ID da foto justamente por não sabermos qual ID será utilizado no endereço. Aliás, "coringa" com "o" é um tipo de pequena vela triangular ou quadrangular. Queremos é um curinga mesmo!

Vamos testar e acessar novamente os dois endereços que acessamos antes?

Ele considera a rota válida, tanto isso é verdade que exibe no console:

```
busca foto com ID 1
busca foto com ID 1
```

Mas ainda temos um problema. Quando eu mudo o ID da URL para 2, a mensagem continua exibindo `busca foto com ID 1`. Precisamos de alguma forma extrair essa informação do ID do endereço acessado. Mais uma vez, o sr.Express pode nos ajudar nessa tarefa. Através de `req.params` podemos ter acesso a todos esses parâmetros da URL:

```
// alurapic/app/routes/foto.js

module.exports = function(app) {

    var api = app.api.foto;

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

    app.get('/v1/fotos/:id', function(req, res) {
        console.log('busca foto com ID ' + req.params.id);
    });
};
```

Revelando o valor do curinga

Você deve estar perguntando como eu sei que é req.params.id, certo? Foi por causa do nosso curinga! Se a nossa URL fosse `v1/fotos/:calopsita`, poderíamos acessar o valor de calopsita como `req.params.calopsita`! Perfeito! Um teste demonstra

que funciona.

Agora que descobrirmos como lidar com URL dinâmicas, já podemos separar o código com nossa lógica do arquivos de rotas. Nosso arquivo de rotas fica assim:

```
// alurapic/app/routes/foto.js

module.exports = function(app) {

    var api = app.api.foto;

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

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

};
```

Na API de fotos, criaremos a função `buscaPorId` com o código que antes estava no arquivo de rotas:

```
// alurapic/app/api/foto.js

var api = {};

api.lista = function(req, res) {

    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' }
    ];

    res.json(fotos);
};

api.buscaPorId = function(req, res) {

    console.log('busca foto com ID ' + req.params.id);

};

module.exports = api;
```

Um teste nunca é de mais para sabermos se o comportamento do nosso nosso servidor continua o mesmo. É realmente, continua funcionando.

Buscando quem procuramos

Agora, no lugar de exibirmos uma simples mensagem no console do servidor, precisamos devolver para um objeto foto no formato JSON! Se já sabemos o ID da foto, o primeiro passo é procurará-la em nossa lista. Podemos fazer isso usando `for`, `forEach` ou `filter`. Porém, vamos usar a função `find` do ECMASCRIPT 6 que facilitará e muito a nossa vida. É claro, ela só funcionará se você instalou a versão do Node.js solicitada no exercício obrigatório de infraestrutura:

```
// alurapic/app/api/foto.js

var api = {};

api.lista = function(req, res) {

    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' }
    ];

    res.json(fotos);
};

api.buscaPorId = function(req, res) {

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

    res.json(foto);
};


module.exports = api;
```

A função `find` recebe como parâmetro uma função de callback que será chamada N vezes para cada item da lista. Em cada chamada, recebemos o elemento na qual ela está operando, foi por isso que usei como nome de parâmetro `foto`, no singular. Dentro da função, pergunto se o ID daquele objeto que está sendo varrido naquele momento é igual ao ID enviado pela requisição. Se for, a função `find` retornará o item da lista e parando, claro, de iterar nos demais elementos porque já encontrou quem queria.

Vamos testar? Ops! Temos uma mensagem de erro no console:

```
ReferenceError: fotos is not defined
```

Isso acontece porque dentro da nossa API `api.buscaPorId` não temos acesso à variável fotos, que é privada em `api.lista`. Podemos resolver isso facilmente, movendo a lista para foto de `api.lista`, ampliando assim seu escopo:

```
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);
};

module.exports = api;
```

Como já disse, sou igual a São Tomé, só acredito vendo. Vamos reiniciar nosso servidor e verificarmos o resultado acessando as URLs:

```
localhost:3000/v1/fotos/1
localhost:3000/v1/fotos/1
```

E se procurarmos por um ID que não existe? A função `.find` retornará `undefined` mas `res.json` se encarregará de não enviar nada.

Tudo certo? Isso significa que podemos selecionar uma foto em nossa página principal que seremos lançados para a tela de edição? Só tentando. Perfeito, dependendo da foto que eu cliquei, seus dados são exibidos na tela de edição.

E agora? Vamos dar uma pausa e praticar com os exercícios!