02

Lembrando do ensino fundamental

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/mean-js/stages/03-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`.

Muito bem, nosso servidor está de pé e compartilhando todo o conteúdo da pasta `alurapic/public`. Isso significa que podemos acessar qualquer arquivo estático dentro desta pasta através do nosso browser, inclusive a `index.html`. Vimos também que esta página, ao ser carregada pelo browser, não exibe qualquer informação de foto e nem poderia.

Nosso servidor precisa ampliar seu vocabulário

Nossa aplicação Angular tenta acessar o recurso `localhost:3000/v1/fotos` para buscar uma lista de fotos, mas este recurso não existe em nosso servidor, indicado pelo código de erro 404. Precisamos criar esse recurso para pelo menos conseguirmos exibir a lista de fotos através da nossa aplicação Angular.

O primeiro passo para criarmos esse recurso é lembrarmos das nossas aulas de português de ensino fundamental. Lá, aprendemos verbos, inclusive a conjugá-los, mas não se preocupe, não haverá conjugação de verbo nesse treinamento, aliviado? Todo verbo indica uma ação e por mais que o browser não tenha sido meu colega de turma, ele também conhece alguns verbos.

Por exemplo, quando digitamos `localhost:3000`, o browser **obtém** a página `index.html` do nosso servidor, que por sua vez não pode ser surdo e ainda ser capaz de conhecer o verbo `obter`. É claro, no mundo da programação usamos termos em inglês o tempo todo, e o verbo usado nessa simples operação de pedido e envio de um recurso é o verbo **GET**, obter. Não é por acaso a especificação do protocolo HTTP definir esse e outros verbos que podem ser usados, mas por enquanto vamos focar apenas no GET.

Então, onde está o problema? Está em nosso servidor, que não está preparado para responder ao verbo GET para o recurso `v1/fotos`, solicitado pela nossa aplicação Angular. É só lembrar que quem obtém, obtém alguma coisa. Que coisa? No caso da nossa aplicação Angular ela quer obter (GET) os recursos `v1/fotos`, por isso usa o verbo GET. Veja que é um nome que não parece nada com um arquivo, inclusive não temos nada dentro de `alurapic/public` parecido com esse nome. E agora?

Precisamos ensinar nosso servidor a responder ao verbo GET (obter), mas para o recurso `v1/fotos`. Isso significa que podemos ter recursos diferentes em nosso servidor para uma mesma ação, GET. É nessa tarefa que o Express nos ajudará.

Para cada verbo HTTP, o Express possui uma função correspondente de mesmo nome. Queremos ligar com o verbo GET para o identificador `v1/fotos`, sendo assim, vamos editar **alurapic/config/express** e adicionar:

```
var express = require('express');
var app = express();

app.use(express.static('./public'));

app.get('/v1/fotos');

module.exports = app;
```

Vamos reiniciar nosso servidor e acessar `localhost:3000/v1/fotos`. Bem, recebemos a mensagem:

```
Cannot GET /v1/fotos
```

Verbo sem ação não existe

É, não foi possível obter o recurso `/v1/fotos`. Faz sentido, porque lá no server, não sabemos qual resposta daremos quando alguém acessar esse recurso. Configuramos essa resposta passando uma função como segundo parâmetro da função `app.get()`:

```
var express = require('express');
var app = express();

app.use(express.static('./public'));

app.get('/v1/fotos', function() {
    console.log('O que você quer?');
});

module.exports = app;
```

Reiniciando e testando. Bem, nosso browser fica aguardando uma resposta eternamente, mas nosso servidor exibe no console "O que você quer?". Temos certeza de que nosso servidor sabe que estamos tentando acessar o recurso, mas apenas imprimimos no console a resposta, não a devolvemos. Aprendemos no capítulo anterior que é através de um fluxo de resposta que podemos responder para o usuário. É por isso que também recebemos `req` e `res` na função:

```
var express = require('express');
var app = express();

app.use(express.static('./public'));

app.get('/v1/fotos', function(req, res) {

    res.end('o que você quer?');
});

module.exports = app;
```

Não sabe onde quer chegar? Trace uma rota!

Reiniciando e testando novamente. Perfeito, recebemos uma resposta! Esse pequeno avanço indica que configuramos uma **rota** no nosso servidor, que devolve uma resposta para quem acessá-la. Você disse uma rota, Flávio? Sim, eu disse. No Express, quando configuramos um recurso, estamos configurando uma rota. Só que queremos uma lista de fotos, não esse texto malcriado.

Vou criar um array Javascript com dois objetos que representam nossa foto. É claro que esses dados devem vir de um banco de dados, mas primeiro vamos focar na integração do Express com o Angular. Mais tarde, aprenderemos a integrar o Express com o MongoDB:

```
var express = require('express');
var app = express();
```

```
app.use(express.static('./public'));

app.get('/v1/fotos', 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' }
    ];
});

module.exports = app;
```

Excelente, só falta devolvermos a resposta. Que tal usarmos o `res.end` , igual fizemos no capítulo anterior?

```
var express = require('express');
var app = express();

app.use(express.static('./public'));

app.get('/v1/fotos', 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.end(fotos);

});

module.exports = app;
```

Reiniciando e... Ops! Recebemos a mesma mensagem de erro, tanto no navegador quanto no servidor:

```
TypeError: first argument must be a string or Buffer
```

O que é o que é? Parece um objeto Javascript mas não é

Isso acontece porque `res.end` espera receber uma String ou um Buffer como parâmetro e o que enviamos é um objeto Javascript. Quando queremos enviar um objeto Javascript que é uma estrutura de dados em memória precisamos convertê-lo para um formato textual chamado JSON. Tinha que ser assim, porque o protocolo http envia apenas texto. Esse formato, como é muito semelhante a estrutura do objeto Javascript pode ser facilmente convertido em objeto quando recebido e vice-versa. Aliás, eles são tão parecidos que muita vezes usamos o termo JSON para nos referirmos ao objeto em memória e sua representação textual. E agora?

Queremos enviar um JSON, certo? Isso significa que temos que transformar o objeto `fotos` em JSON para podeis enviá-lo como String. Porém, não faremos isso manualmente. Usaremos uma função do fluxo de resposta especializada em converter objetos Javascript no formato JSON e enviá-lo de uma só vez. É por isso que existe a função `res.json` :

```
var express = require('express');
var app = express();

app.use(express.static('./public'));

app.get('/v1/fotos', 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);
});

module.exports = app;
```

Um ponto interessante, é que nossa aplicação em Angular recebe os dados no formatado JSON, mas tanto $http e $resource convertem essa estrutura textual em objeto Javascript sem nos preocuparmos. Mais uma vez, esse é o motivo de muitos desenvolvedores usarem o termo JSON para um objeto em memória e para a representação textual desse objeto.

Mas vamos voltar para nosso teste e verificar se tudo funciona. Acessando a URL `localhost:3000/v1/fotos` recebemos nosso JSON com tudo que tem direito!

```
[{"titulo":"Leão","url":"http://www.fundosanimais.com/Minis/leoes.jpg"},{"titulo":"Leão 2","url":"ht
```

E agora? O que acontecerá se abrirmos o endereço `localhost:3000` ? Sabemos que a página `index.html` será carregada e com ela a nossa aplicação Angular, que realizará uma requisição do tipo GET para `v1/fotos` . Um novo teste demonstra que funciona. Perfeito!

Aprendemos até agora a levantar um servidor web, compartilhar arquivos estáticos, inclusive a criar respostas de acordo com o verbo e identificador enviado para o servidor, tudo com o auxílio do Express. Será que podemos continuar a avançar ou ainda podemos melhorar algo? Cenas do próximo capítulo.