03

A melhor foto é a do JSON

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/mean-js/stages/07-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`.

Nosso servidor, de um simples bebezinho está cada vez mais se tornando um homenzarrão, mas ele precisa comer muito angu para suprir todas as necessidades que nossa aplicação Angular necessita. Uma que ainda falta implementarmos é o cadastro de novas fotos.

De braços abertos para o JSON

Na tela principal da aplicação, aquela que exibe nossa lista de fotos, há o botão `Nova foto` que ao ser clicado dispara uma rota do lado do cliente feita em Angular que carregará a view parcial "foto.html`. Mas é claro, você fez o treinamento de Angular antes de fazer este e já sabe disso, mas não custa nada lembrar. Quando preenchermos todos os dados do formulário, nossa aplicação Angular submeterá um JSON contendo os dados da foto para nosso server.

Precisamos criar uma rota que saiba receber esse JSON, atualizar nossa lista de fotos e devolver uma resposta. Já fixou que arquivos de rotas ficam em `alurapic/app/routes`? Como é uma rota que diz respeito à fotos, vamos editar `alurapic/app/routes/fotos.js`:

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

module.exports = function(app) {

    var api = app.api.foto;

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

    app.route('/v1/fotos/:id')
        .get(api.buscaPorId)
        .delete(api.removePorId);

      // qual rota usaremos para adicionar uma foto?
};
```

E agora, qual rota usaremos para adicionarmos uma foto? Não faz sentido usarmos `/v1/fotos/:id` porque não enviaremos o ID da foto, mas um objeto foto no formato JSON! Que tal usarmos `/v1/fotos` que já usamos para ligar com o verbo GET, mas dessa vez, planejaremos uma reposta quando o verbo for POST. É comum o verbo POST ser usado toda vez que desejamos criar um novo recurso no servidor. Inclusive, podemos usar a mesma estratégia com `app.route` que usamos no capítulo anterior. Nosso arquivo fica assim:

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

    var api = app.api.foto;

    // mudando a declaração da rota e adicionando suporte ao verbo POST
    app.route('/v1/fotos')
```

```
        .get(api.lista)
        .post(api.adiciona);

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

Agora, precisamos implementar nossa API:

```
// alurapic/app/api/foto.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) {

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

    res.sendStatus(204);
};

// novidade aqui

api.adiciona = function(req, res) {

    // o que adicionar? Onde está a foto enviada?
};

module.exports = api;
```

Cabeça, corpo e membro: onde está o JSON?

Sabemos pegar parâmetros de URL's, mas um JSON não é enviado como um parâmetro, ele é enviado no "corpo" da mensagem. Sendo assim, precisamos de alguma maneira acessar esse "corpo" através do objeto que representa nosso fluxo de requisição.

```
// alurapic/app/api/foto.js
// código anterior omitido

api.adiciona = function(req, res) {

    var foto = req.body;
    console.log(foto);
};
// código posterior omitido
```

É através de `req.body` que acessamos o JSON enviado pela aplicação Angular. Que tal cadastramos uma foto e vermos sendo exibido no console aquela foto que foi enviada pela nossa aplicação Angular? Ops! É impresso `undefined` :

```
consign v0.1.2 Initialized in app
+ ./api/foto.js
+ ./api/grupo.js
+ ./routes/foto.js
+ ./routes/grupo.js
Servidor iniciado
undefined <-- não era o que esperávamos
```

Transplante de JSON

O problema é que ninguém adicionou o JSON enviado convertido para objeto na propriedade `req.body` . Precisamos pedir que o Express faça isso para nós através de um middleware, isto é, alguém que filtrará nossas requisições e quando achar um JSON no corpo da mensagem o converterá para objeto Javascript e o armazenará na propriedade `req.body` . O middleware que faz isso é o [body-parser (https://github.com/expressjs/body-parser)](https://github.com/expressjs/body-parser). Sua instalação é feita no terminal através do npm:

```
npm install body-parser@1.14.1 --save
```

Até agora só tínhamos configurado um middleware no Express, aquele que torna nossa pasta `alurapic/public` acessível. Vamos alterar `alurapic/config/express.js` e adicionar o `body-parser` :

```
// alurapic/config/express.js

var express = require('express');
var consign = require('consign');
var bodyParser = require('body-parser'); // importou o módulo
var app = express();

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

// configurando o middleware body-parser
app.use(bodyParser.json());

consign({cwd: 'app'})
```

```
    .include('api')
    .then('routes')
    .into(app);

module.exports = app;
```

Como aprendemos, adicionamos novos middlewares ao Express através da função `app.use`. Além disso, pedimos ao `body-parser` que faça o *parser* de qualquer JSON que for recebido pelo servidor. Inclusive ele converterá o JSON para um objeto Javascript que pode ser acessado através de `req.body`.

Vamos reiniciar o servidor e testar.

```
+ ./api/foto.js
+ ./api/grupo.js
+ ./routes/foto.js
+ ./routes/grupo.js
Servidor iniciado
{ titulo: 'Gato Guerreiro',
  url: 'https://upload.wikimedia.org/wikipedia/commons/9/98/Gato_bombay.jpg',
  descricao: 'Não subestime este gato!',
  grupo: 3 }
```

Agora, quando gravarmos um foto em nossa aplicação Angular, o JSON da foto é enviado para nosso servidor através do verbo POST e nosso server consegue exibir a foto enviada no terminal. Para concluir, basta adicionar a foto em nossa lista de fotos. Como resposta, enviaremos a foto de volta, mas com seu ID preenchido. Aliás, como geraremos o ID da nossa foto? Vamos criar uma variável chamada `CONTADOR_ID` que começa com valor 2. Para cada foto adicionada iremos incrementar seu contador e adicionar o próximo ID na foto. Isso é temporário, até fazermos a integração com o banco:

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

api.adiciona = function(req, res) {

    var foto = req.body;
    foto._id = ++CONTADOR_ID;
    fotos.push(foto);
    res.json(foto);
};
```

Prontinho! Podemos cadastrar fotos e ainda selecioná-las da lista de fotos da aplicação visualizando-a novamente. Só falta implementarmos a alteração de fotos, assunto do próprio capítulo.