# **O que são os verbos HTTP?**

Neste capítulo vimos o conceito de verbos HTTP e até tiramos proveito deles para implementarmos algumas funcionalidades. Mas o que são os verbos HTTP? Para que eles servem? escreva com suas palavras o que você entendeu deles.

## **Resposta do instrutor**

Os verbos HTTP definem as possíveis operações que podem ser feitas sobre o protocolo HTTP. Eles formam um conjunto pequeno de interfaces bem definidas, que facilita a bastante a maneira como dois recursos trocam informações utilizando o HTTP, visto que eles precisam apenas utilizar esses métodos de acordo com a semântica de cada operação, sem que precisem sempre implementar tudo do zero. Essa padronização cria uma uniformidade muito importante para o protocolo HTTP.

Se você quiser aprofundar um pouco mais nesse assunto (o que parece uma excelente ideia para quem pretende trabalhar com serviços web), você pode começar dando uma olhada nesse post da Caelum, que traz um apanhado bem legal de características do padrão REST (que é todo baseado no funcionamento do HTTP):

<https://www.caelum.com.br/apostila-vraptor-hibernate/rest/#11-3-o-triangulo-do-rest>

# **Onde mais tirar proveito dos verbos HTTP?**

Utilizamos os conceitos do HTTP para definir as rotas da aplicação como sendo uma combinação de url + verbo. Fizemos isso para diferenciar o GET e o POST de Produtos, por exemplo. Em que outra situação esse conhecimento dos verbos HTTP poderia ser utilizado?

## **Resposta do instrutor**

Uma outra aplicabilidade dentro do próprio sistema da Casa do Código poderia ser para fazer a exclusão de um registro já cadastrado. Para essa operação poderia ser utilizado o método DELETE. Pesquise um pouco e tente implementar essa operação!

# **Criando o formulário de cadastro de produtos**

Neste capítulo escrevemos todo o código necessário para implementar o cadastros dos produtos no nosso sistema da casa do código. Para isso alguns arquivos precisaram ser criado e outros alterados. Agora é a sua vez de fazer o mesmo em sua aplicação. =D

Comece criando o html para o formulário na pasta app/views/produtos e você pode chamá-lo de form.ejs, por exemplo. Deve ser um html bem simples, com um form que tem uma action para “produtos/salva”, três campos com suas respectivas labels para que o usuário informe o título, a descrição e o preço do produto e por fim o submit com o value “Gravar”.

Após criar o form, você precisa criar uma rota para que este form fique acessível a partir da urlhttp://localhost:3000/produtos/form.

Para isso, configure então a nova rota no seu arquivo de configuração de rotas e escreva a function de callback dessa rota fazendo o render do novo arquivo produtos/form.ejs que você acabou de criar.

Após a configuração teste no navegador se a nova rota está funcionando e compartilhe seu código aqui! E se antes de configurar a rota no produtos.js você tentar acessar diretamente o caminho do novo form criado direto no navegador, o que acontece?

## **Resposta do instrutor**

O arquivo form.ejs que contem o html para o formulário de cadastro dos produtos, deve ficar assim:

<html>  
 <body>  
 <form action="/produtos/salva" method="post">  
 <div>  
 <label for="titulo">Título</label>  
 <input type="text" id="titulo" name="titulo" placeholder="Informe o titulo do Livro"/>  
 </div>  
 <div>  
 <label for="preco">Preço</label>  
 <input type="text" id="preco" name="preco" placeholder="120.05"/>  
 </div>  
 <div>  
 <label for="descricao">Descrição</label>  
 <textarea cols="20" rows="10" id="descricao" name="descricao" placeholder="Fale sobre o livro"></textarea>  
 </div>  
  
 <input type="submit" value="Gravar"/>  
 </form>  
 </body>  
</html>

Para conseguir agora que esse novo form fique acessível para o usuário é preciso que seja criada uma configuração de rota para ele, pois os arquivos criados dentro da pasta views do EJS não ficam disponíveis para acesso direto pelo cliente da aplicação. Essa é a forma que o EJS trabalha.

A configuração da nova rota com sua respectiva função controller no produtos.js fica assim:

app.get("/produtos/form",function(req, res) {  
 res.render('produtos/form');  
 });

# **Configuração do body-parser**

Para que o parse do body da requisição funcione precisamos ensinar o express como fazê-lo. Para isso utilizamos a lib body-parser.

Instale essa nova lib no seu projeto utilizando o node package manager, lembrando de passar o parâmetro que guarda essa informação como uma nova dependência do projeto.

Agora vá até o arquivo de configurações do express, carregue essa nova lib através require e guarde-a numa variável. Use essa variável para informar ao express que agora ele deve usá-la em seu ciclo de vida para fazer o parse de urlencoded.

Aproveite para já passar o parâmetro informando que você está usando a versão extended, evitando assim a mensagem de deprecated.

## **Resposta do instrutor**

Para instalar o body-parser:

npm install body-parser --save

Agora é só carregar o módulo no express.js e passá-lo para a function use do express, informando que ele agora é o responsável por fazer parse de urlencoded. E sem esquecer de passar o parâmetro extended: trueque evita a mensagem de deprecated:

var app = require('express')();  
var load = require('express-load');  
  
var bodyParser = require('body-parser');  
  
module.exports = function() {  
  
 var app = express();  
 app.set('view engine', 'ejs');  
 app.set('views','./app/views');  
  
 app.use(bodyParser.urlencoded({extended: true}));  
  
 load('routes',{cwd: 'app'})  
 .then('infra')  
 .into(app);  
  
 return app;  
  
}

Lembre que para o código funcionar é muito importante que os novos módulos carregado no express apareçam em uma ordem que faça sentido. O uso do body-parser deve ser informado antes do carregamento das rotas para que quando elas forem carregadas, já tenham a informação de como deverá ser feito o parse dos seus forms.

# **Novo método salva() no ProdutoDAO**

Antes de configurar o controller do cadastro do produto, é preciso que ProdutosDAO esteja pronto para executar a ação de salvar. Para isso criamos o novo método salva() nesta classe e o atribuímos ao seuprototype.

Faça isso também na sua aplicação. Crie o método salva() no ProdutosDAO. Esse método deve ser implementado por uma function que recebe como parâmetros o produto a ser salvo e a função de callback a ser executada.

Dentro dele, invoque a query a partir da connection passando a String necessária para escrever um insert para o MySQL e aproveite a facilidade que o driver MySQL do javascript provê de já transformar o json recebido nos valores correspondentes de cada respectiva coluna no banco de dados.

Você pode fazer isso utilizando a função SET ? e passando a variável que contem o json do produto como parâmetro para a função query, além é claro da função de callback.

O que achou de escrever o insert dessa maneira? De que outra forma você poderia ter feito essa conversão dos dados, caso não tivesse um json exatamente compatível com o banco, por exemplo?

## **Resposta do instrutor**

A nova função salva no ProdutosDAO fica assim:

ProdutosDAO.prototype.salva = function(livro,callback) {  
 this.\_connection.query('INSERT INTO livros SET ?', livro, callback);  
}

Uma outra maneira de passar os parâmetros para a query seria utilizando o values exatamente como se faz na sql pura e passar uma ‘?’ no valor de cada parâmetro da query. Passando depois também cada valor como parâmetro da função query no javascript.

# **Criando a rota para a action do botão**

Após criar a rota para exibir o form de cadastro, precisamos também configurar uma nova rota para a receber a requisição disparada pela action do form. Para isso implementamos a rota para “produtos/salva” e mapeamos o método POST do HTTP para ela, já que esse é o método utilizado no form.

Implemente agora essa rota no seu projeto, fazendo com que dentro da sua controller haja uma variável “produto” para receber o corpo da requisição, seja criada uma nova conexão com o banco e instanciado um novo objeto de ProdutoDAO, que será utilizado para invocar o método salva, passando o produto que foi recebido do form. No fim, esse controller deve renderizar de volta a view “produtos/lista”.

Faça o teste da nova rota, cadastrando um novo produto no formulário e clicando no botão gravar. O que acontece na tela após o clique? E no banco, o novo produto foi salvo?

## **Resposta do instrutor**

No arquivo produto.js, a configuração da nova rota que mapeia o método post para a ulr ‘/produtos/salva’ com sua respectiva função controller deve ficar da seguinte maneira:

app.post("/produtos/salva",function(req,res) {  
 var produto = req.body;  
 console.log(produto);  
  
 var connection = app.infra.connectionFactory();  
 var produtosDAO = new app.infra.ProdutosDAO(connection);  
  
 produtosDAO.salva(produto,function(erros,resultado){  
 res.render("produtos/lista");  
 });  
  
 connection.end();  
  
 });

Essa função vai funcionar perfeitamente para inserir os dados no banco, porém falha em redirecionar a response de volta para a view de listagem de produtos porque essa view espera um json com a lista de produtos que não está sendo passando. Essa situação será corrigida na próxima aula.

## **Sua resposta**

Resolvendo o redirecionamento

**app/routes/produtos.js**

// código anterior omitido  
 app.post('/produtos', function(req, res) {  
 var connection = app.infra.connectionFactory();  
 var produtosDAO = new app.infra.ProdutosDAO(connection);  
 produtosDAO.salva(req.body, function(erros, resultados) {  
 res.redirect('/produtos');  
 });  
 });  
}