# **O que é o Content Negotiation?**

Nesse capítulo falamos sobre o conceito de Content Negotiation. Escreva aqui o que você entendeu desse conceito.

## **Resposta do instrutor**

Content Negotiation é um mecanismo definido no HTTP que torna possível servir diferentes formatos de um mesmo conteúdo a partir da mesma url. Evitando assim que se precisasse criar novas urls para cada formato necessário para exibir a lista de produtos, por exemplo.

Ele funciona através do Header Accept, em que o cliente que está consumindo a url informa qual tipo de dados, ele aceita receber no response de sua requisição. Os navegadores tradicionais, por exemplo, utilizam por default o valor “text/html” em seu Accept.

# **Onde mais utilizar o Content Negotiation?**

Nós utilizamos o recurso do Content Negotiation para atender a requisição à url de produtos, retornando o formato de dados específico de acordo com o que o cliente pedir. Em que outras situações que você já passou, precisou ou poderia ter utilizado o Content Negotiation? Ou mesmo no próprio sistema da Casa do Código que estamos implementando, você vê alguma outra situação em que ele poderia ser útil?

## **Resposta do instrutor**

Praticamente todas as urls públicas de sistemas escritos em Node são boas candidatas a suportar o envio de sua resposta em mais de um tipo de formato, visto que o Node é pensado para aplicações em servidor, que muitas vezes acabaram sendo implementadas na forma de serviços REST. Por esse motivo é bem comum que tenha diferentes tipos de clientes que trabalham melhor com diferentes tipos de dados.

# **Implementando o Content Negotiation**

Para poder atender a mais de um tipo de dados, alteramos função controller da rota que lista os produtos para que ela passe a responder o conteudo em um formato diferente de acordo o que está sendo passado no Header Accept da requisição.

Faça essa alteração na sua controller: utilize o res.format() para identificar qual o formato aceito e envie a response de acordo com ele.

Para verificar o funcionamento crie um arquivo simples e utilizando somente o módulo http do próprio Node e use-o para simular requisições vindas de algum outro sistema, hora pedindo um json, hora pedindo um html.

Compartilhe seu código com a gente. De que outra maneira você conseguiria testar esses cenários caso não quisesse criar uma arquivo somente para isso?

## **Resposta do instrutor**

O controller da rota do GET de ‘/produtos’ agora fica assim:

app.get("/produtos",function(req, res) {  
  
 var connection = app.infra.connectionFactory;  
 var produtosBanco = new app.infra.produtosBanco(connection);  
  
 produtosBanco.lista(function(err, results){  
  
 res.format({  
 html: function(){  
 res.render("produtos/lista",{lista:results});  
 },  
 json: function(){  
 res.json(results);  
 }  
 });  
  
 });  
  
 connection.end();  
  
 });

O arquivo criado para simular uma requisição vinda de um sistema android foi chamado de cliente-android.js e ficou assim:

var http = require('http');  
  
var configuracoes = {  
 hostname: 'localhost',  
 port: 3000,  
 path: '/produtos',  
 headers: {  
 'Accept': 'application/json'  
 }  
};  
  
http.get(configuracoes, function(res){  
 console.log(res.statusCode);  
 res.on('data', function(body){  
 console.log('Corpo:' +body);  
 });  
});

Para executá-lo basta utilizar o comando do node já conhecido para executar um arquivo. Você pode alterar o seu Header Accept e fazer vários testes para ver se o seu servidor está de fato entregando o conteudo conforme requisitado.

Outras alternativas para fazer esses testes seriam utilizar o comando CURL presente em terminais de sistemas operacionais como MAC e Linux, ou utilizar algum plugin de navegador que permita manipular dados do HTTP nas requisições.

Pesquise sobre essas possibilidades. Quando se aprende uma nova tecnologia, é sempre uma boa prática estudar além do conteudo básico.

# **Implementando suporte a JSON no cadastro**

Para tornar possível que a aplicação receba os dados do formulário de cadastro de produto também no formato JSON, precisamos ensinar o express a fazer parse desse formato também através do body-parser.

Faça essa alteração também no seu código adicionando mais este middleware no arquivo de configuração do express. O seu uso é bem semelhante ao que fizemos para suportar o parse do urlencoded.

Para testar essa nova funcionalidade, crie uma arquivo para fazer POST na url “/produtos” passando algum produto como parâmetro. Ou utilize alguma outra maneira de fazer POST que você conheça.

Execute seus testes e veja se funciona.

Qual foi o Status Code recebido após a requisição? O produto realmente foi cadastrado no banco?

## **Resposta do instrutor**

A única alteração de código necessária nesse exercício é adicionar o uso do parse de json via body-parser ao objeto do express. Essa alteração foi feita arquivo express.js logo após a linha que adiciona o parse deurlencoded:

app.use(bodyParser.urlencoded({extended: true}));  
 app.use(bodyParser.json());

Para fazer o teste via arquivo javascritpt, criamos o cadastra-livro-terminal.js:

var http = require('http');  
  
var configuracoes = {  
 hostname: 'localhost',  
 port: 3000,  
 path: '/produtos',  
 method: 'post'  
 headers: {  
 'Accept': 'application/json',  
 'Content-type': 'application/json'  
 }  
};  
  
var client = http.request(configuracoes, function(res){  
 console.log(res.statusCode);  
 res.on('data', function(body){  
 console.log('Corpo:' +body);  
 });  
});  
  
var produto = {  
 titulo : 'mais sobre o node',  
 descricao : 'node, javascript e um pouco de http',  
 preco : 29.50  
}  
  
client.end(JSON.stringify(produto));

A grande novidade nesse arquivo com relação ao criado exercício anterior é que agora precisamos guardar a função http.request() numa variável que chamamos de client. Para depois invocar a funçãoclient.end() que é quem de fato envia a requisição.

Além disso, criamos um JSON com os dados do produto a ser cadastrado e o passamos como parâmetro do POST, ou seja como body da requisição utilizando a função JSON.stringify() que faz o trabalho de converter o JSON em String, que é o formato de dados que o HTTP trafega.

O último ponto importante foi adicionar o header 'Content-type': 'application/json’ que serve para informar para o servidor em qual formato de dados os parâmetros do form estão sendo enviado.