01

Serviços Web -REST

Transcrição

Já estudamos como o modelo de requisição-resposta funciona usando o browser para esse estudo. Mas será que toda a requisição `HTTP` sempre tem como origem um navegador? E toda resposta só possui conteúdo que ele entende: `HTML, CSS, Javascript e imagens` ?

Nesse capítulo veremos como as aplicações conseguem se comunicar e responder as questões levantadas anteriormente.

##Serviços WEB

Hoje existem milhões de softwares rodando ou sendo desenvolvidos em várias linguagens de programação e frameworks. Tais softwares não vivem necessariamente isolados e podem querer se comunicar de alguma forma.

Um exemplo clássico é o login via rede social que estamos cada vez mais habituados. Essa conversa acaba sendo transparente para nós, usuários, já que exige uma autorização de acesso às nossas informações.

Em outros momentos as aplicações conversam sem que nada visual seja implementado ou mesmo uma autorização do cliente seja pedida.

As aplicações que disponibilizam serviços para outras são chamadas de *webservices*. E uma *API* de utilização é documentada para uma integração eficiente entre sistemas.

Temos serviços web para trabalhar com pagamentos(*Paypal* é um exemplo famoso), upload de imagens, transformação de CEP em endereços textuais e diversos outros. Tudo isso é feito através do poderoso protocolo **HTTP**.

##Cenário de trabalho

Você muito provavelmente já teve uma péssima experiência quando estava sem conexão com a internet usando um aplicativo móvel. Alguns apps não funcionam sem um acesso a rede porque as principais funcionalidades são feitas via requisições **HTTP**.

Essas requisições são implementadas programaticamente pelo desenvolvedor. Podemos implementá-las em várias linguagens de programação: `Java, PHP, Javascript etc.`

Para exemplificar vamos imaginar que temos aqui na *Alura* uma divisão responsável por uma aplicação de pedido de refeições que é a *AluraFood*.

A *AluraFood* tem duas equipes em ação: a do serviço web(ou simplesmente *API* web) e a dos apps mobile(*Android* e *iOS*).

Os desenvolvedores responsáveis pela tela de listagem de restaurantes vão precisar receber do serviço os detalhes de cada restaurante. Felizmente o pessoal responsável pelo *webservice* já documentou exatamente o que seria necessário:

```
Listagem de todos os restaurantes --> GET - http://alurafood.com/api/restaurante
```

Como retorno dessa requisição poderíamos receber o seguinte conteúdo `HTML` :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AluraFood</title>
    <link rel="stylesheet" type="text/css" href="principal.css">
    <script type="text/javascript" src="build.js"></script>
</head>
<body>
    <table>
        <tr>
            <td>Bob's</td>
            <td>8</td>
            <td>R. do México 100</td>
            <td><img src="http://alurafoods.com/uploads/logos/bobs.png"/></td>
        </tr>
        <tr>
            <td>Subway</td>
            <td>8.5</td>
            <td>Av. Rio Branco 202</td>
            <td><img src="http://alurafoods.com/uploads/logos/subway.png"/></td>
        </tr>
        <tr>
            <td>Experimenta Lanches</td>
            <td>9</td>
            <td>R. do Brasil 545</td>
            <td><img src="http://alurafoods.com/uploads/logos/e-lanches.png"/></td>
        </tr>
    </table>
</body>
</html>
```

Perceba que temos uma listagem de restaurante sendo apresentada dentro de uma tabela(elemento **table** do HTML) e cada linha(elemento **tr**) possui 4 colunas(**td**). Dentro de cada coluna temos as informações dos restaurantes: `nome, nota de avaliação, endereço e logo .`

Logo percebemos que os responsáveis precisarão realizar uma análise do conteúdo `HTML` e extrair dele somente as informações necessárias. Esse ato de analisar o documento é chamado de realizar um parsing do arquivo.

Perceba que o `HTML` tem muito mais do que o necessário para essa equipe, é um formato de verbosidade considerável: temos cabeçalhos e tags diversas. Para piorar estamos trafegando muito mais informações do que o necessário e onerando até mesmo a banda do nosso usuário. Péssimo cenário não é mesmo?

Pensando nessa deficiência do `HTML` temos outros formatos que fazem mais sentido quando uma representação de um recurso (um restaurante) se faz necessário. Temos como exemplo mais legível o **XML(eXtensible Markup Language)** que poderia ser devolvido como resposta e ter o seguinte conteúdo:

```
<?xml version="1.0"?>
<restaurantes>
    <restaurante>
        <nome>Bob's</nome>
```

```
        <avaliacao>8</avaliacao>
        <endereco>R. do México 100</endereco>
        <logo>http://alurafoods.com/uploads/logos/bobs.png</logo>
    </restaurante>
    <restaurante>
        <nome>Subway</nome>
        <avaliacao>8.5</avaliacao>
        <endereco>Av. Rio Branco 202</endereco>
        <logo>http://alurafoods.com/uploads/logos/subway.png</logo>
    </restaurante>
    <restaurante>
        <nome>Experimenta Lanches</nome>
        <avaliacao>9</avaliacao>
        <endereco>R. do Brasil 545</endereco>
        <logo>http://alurafoods.com/uploads/logos/e-lanches.png</logo>
    </restaurante>
</restaurantes>
```

Outro famoso formato e onerando menos ainda a rede, por ser mais leve, é o **JSON(JavaScript Object Notation)**:

```
[
   {
      "nome": "Bob's",
      "avaliacao": "8",
      "endereco": "R. do México 100",
      "logo": "http://alurafoods.com/uploads/logos/bobs.png"
   },
   {
      "nome": "Subway",
      "avaliacao": "8.5",
      "endereco": "Av. Rio Branco 202",
      "logo": "http://alurafoods.com/uploads/logos/subway.png"
   },
   {
      "nome": "Experimenta Lanches",
      "avaliacao": "9",
      "endereco": "R. do Brasil 545",
      "logo": "http://alurafoods.com/uploads/logos/e-lanches.png"
   }
]
```

Mas como especificar à aplicação de serviço que gostaríamos de receber em um formato `JSON` ? Via cabeçalho **HTTP**!

Para indicar que queremos resposta no formato `JSON` usa-se um **Accept: application/json** como cabeçalho HTTP. Por outro lado já na resposta uma indicação desse conteúdo é especificado pelo cabeçalho **Content-Type: application/json**.