

REST/JSON – Introdução REST/JSON com Spring

Sergio Bonato – SIN e CCP – ARQ SW – USJT - 2018

REST

- REST, ou Transferência de Estado Representativo (REpresentational State Transfer) é uma maneira simples de realizar interações em sistemas independentes. Sua popularidade vem crescendo desde 2005, o que inspira o design de serviços, como por exemplo a API do Twitter. Isso está relacionado ao fato do REST permitir a interação sem stress com diversos clientes, tanto em dispositivos móveis quanto qualquer outro website/serviço.
- Na teoria, REST não tem nada a ver com web, mas o conceito é amplamente utilizado na área, e foi inspirado pelo HTTP. Como resultado, onde existe HTTP, o REST pode ser utilizado.

REST (continuação)

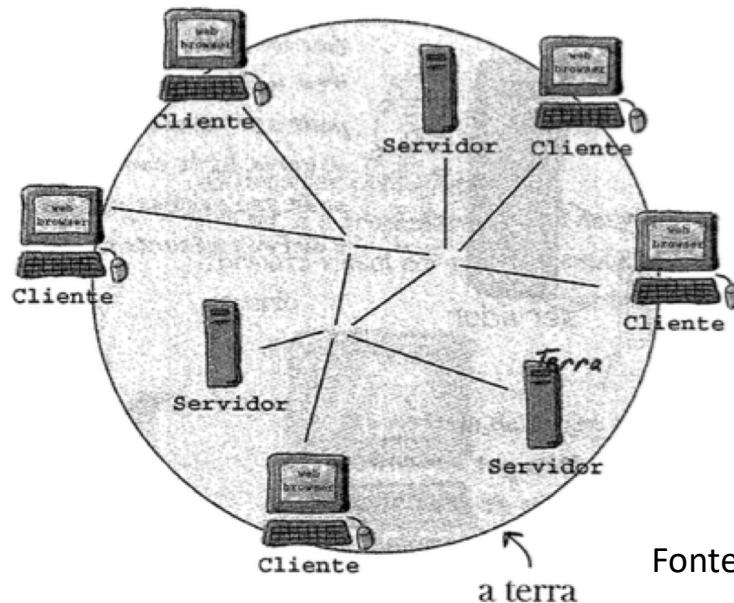
- A alternativa é criar convenções relativamente complexas em cima do HTTP. As vezes, isso nos leva a novas linguagens baseadas no XML. O melhor exemplo disso é o SOAP.
- Você precisa aprender por completo um conjunto de novas convenções, mas você nunca vai utilizar o HTTP com todos os seus recursos.
- Como o REST foi inspirado pelo HTTP e utiliza sua base, é melhor entendermos primeiro como o HTTP de fato funciona.

O Protocolo HTTP conceitos básicos

A arquitetura básica web é composta por 3 elementos:

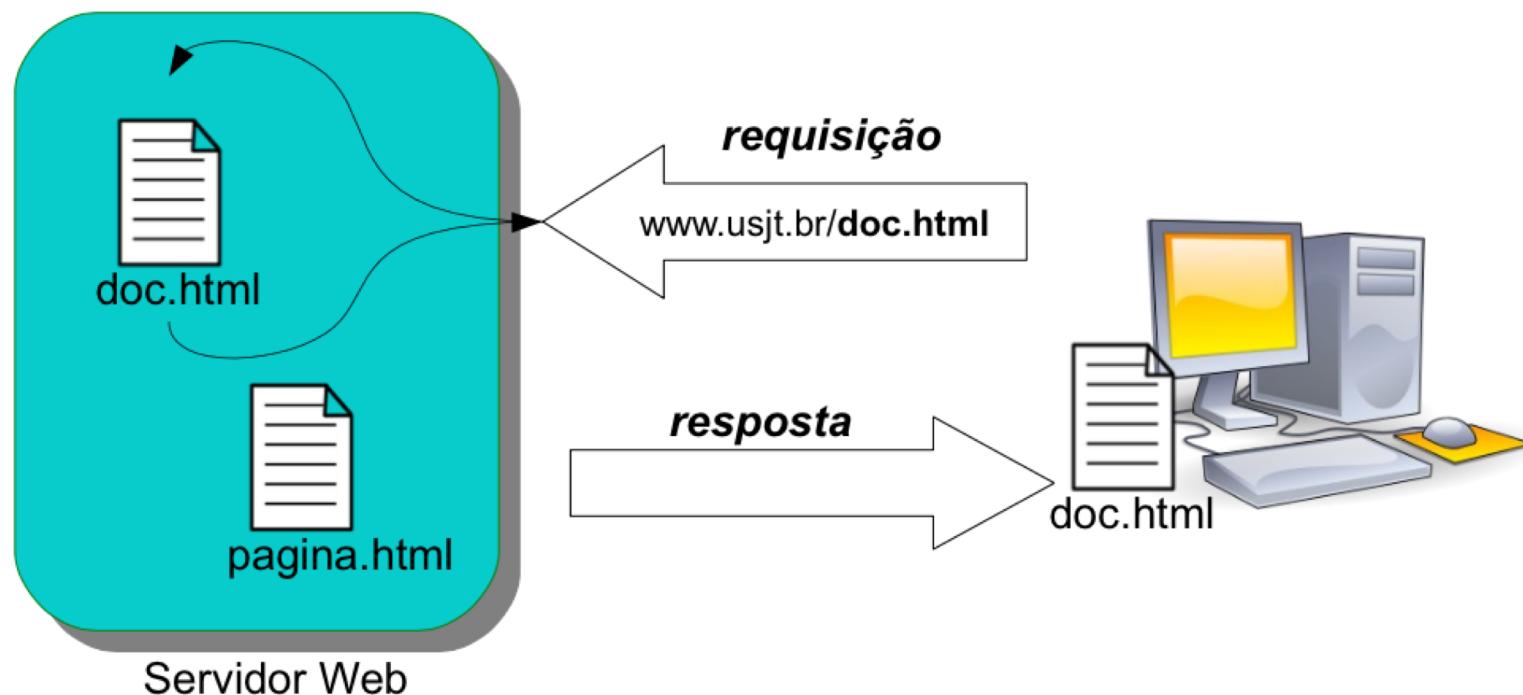
- Servidor Web
- Cliente (navegador web entende HTML)
- Protocolo de comunicação HTTP

A web consiste em zilhões de clientes (usando browsers como o Mozilla ou o Safari) e servidores (rodando aplicações como o Apache), conectados através de redes com fio e wireless. Nossa objetivo é construir uma aplicação que os clientes ao redor do mundo possam acessar. E nos tornarmos estupidamente ricos.

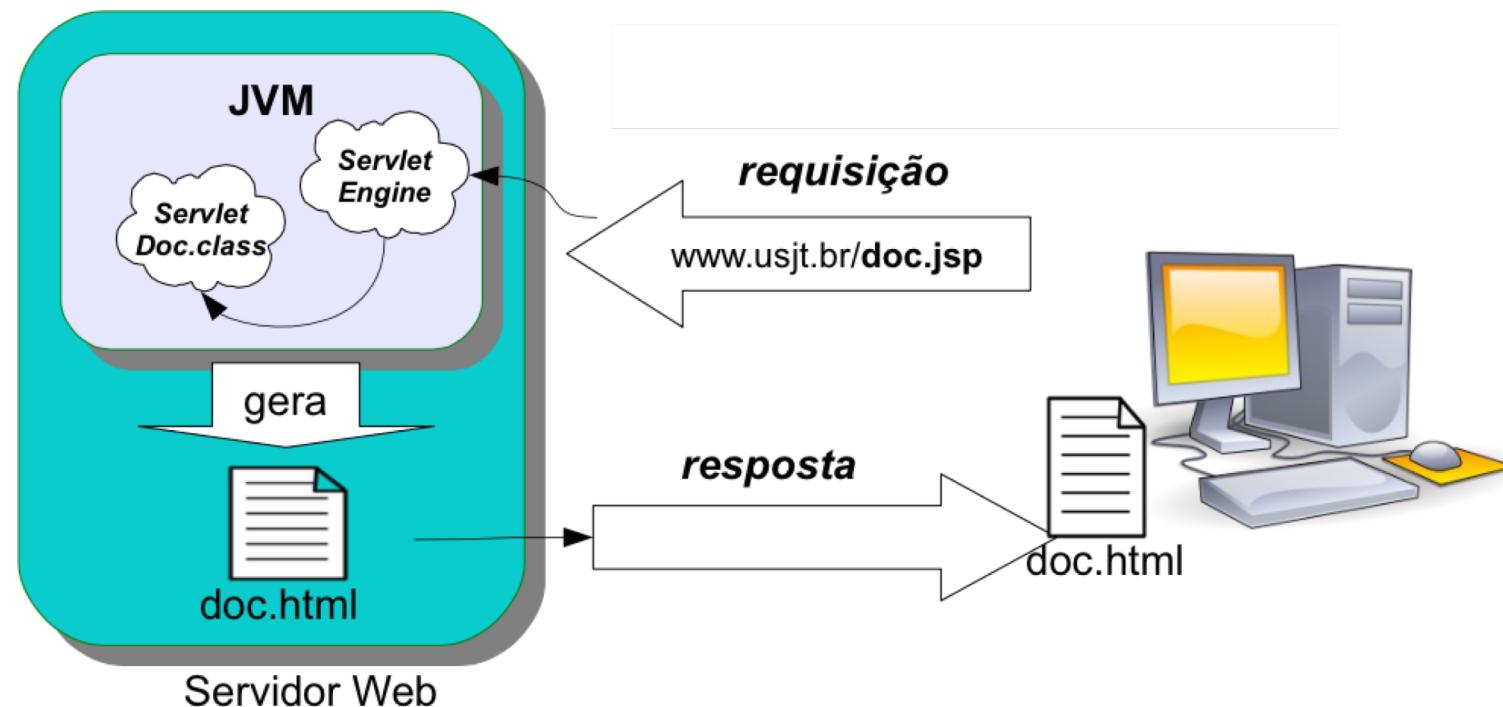


Fonte: (BASHAM, SIERRA, BATES 2008), p. 3

Servidor Web – Conteúdo Estático



Servidor Web – Conteúdo Dinâmico



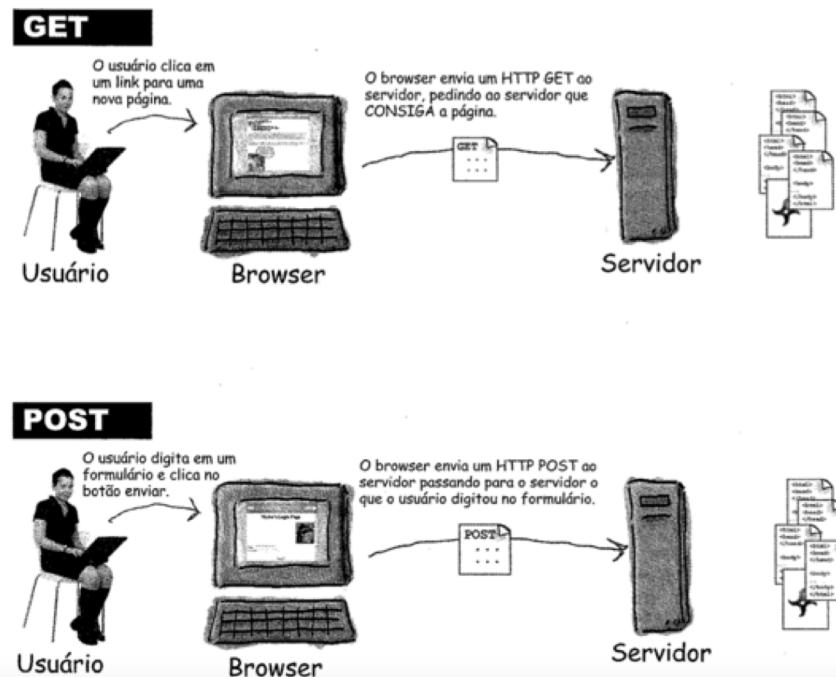
Protocolo HTTP – Cliente Servidor e Stateless

- Um protocolo é uma convenção ou padrão que controla conexão, comunicação ou transferência de dados; são regras que definem o formato das mensagens trocadas.
- O protocolo HTTP é utilizado para comunicação entre cliente (navegador) e servidor Web. É baseado no modelo **cliente/servidor** ou **requisição/resposta** (request/response).
- O HTTP é **stateless**, isto é, a princípio, nenhuma requisição é mantida no servidor.

Protocolo HTTP – Requisição

- Uma requisição consiste no envio de um pacote de dados HTTP solicitando ao servidor um determinado recurso (html, jsp, imagem, etc.). Deve conter um comando ou método que diz o que o servidor Web deverá fazer com a requisição.

Exemplos de métodos HTTP
Fonte: (BASHAM, SIERRA, BATES 2008), p. 12



Formato da Requisição HTTP GET

Em uma solicitação GET, os parâmetros (se existir algum) serão anexados à primeira parte da solicitação URL, iniciando-se por uma "?". Os parâmetros são separados usando-se o "&".

O Método HTTP.

A linha de Solicitação.

O caminho para o recurso no servidor.

A versão do protocolo que o browser está solicitando.

Os headers da Solicitação.

```
GET /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;
rv:1.4) Gecko/20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/
gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Fonte: (BASHAM, SIERRA, BATES 2008), p. 15

Formato da Requisição HTTP POST

A linha de Solicitação.

O Método HTTP.

O caminho para o recurso no servidor.

A versão do protocolo que o browser está solicitando.

Os headers da Solicitação.

O corpo da mensagem, algumas vezes chamado de "payload".

Destas vez os parâmetros estão aqui, no final do corpo e, portanto, não ficam limitados da maneira que ficariam quando se usa um GET, e precisam ser colocados na linha de Solicitação.

```
POST /advisor/selectBeerTaste.do HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
color=dark&taste=malty
```

Fonte: (BASHAM, SIERRA, BATES 2008), p. 16

Resposta: a resposta é um pacote de dados HTTP, formatado como HTML. O conteúdo HTML vem dentro do protocolo HTTP.

A versão do protocolo que o servidor web está usando.

O código de status HTTP para a Resposta.

Uma versão texto do código de status.

HTTP/1.1 200 OK

Set-Cookie: JSESSIONID=0AAB6C8DE415E2E5F307CF334BFCA0C1;
Path=/testEL

Content-Type: text/html

Content-Length: 397

Date: Wed, 19 Nov 2003 03:25:40 GMT

Server: Apache-Coyote/1.1

Connection: close

headers de Resposta HTTP.

O valor do header de resposta para o content-type é conhecido como MIMÉ type. O MIMÉ type informa ao browser que tipo de dado o browser está para receber, para que este possa saber como processá-lo.

O corpo traz o HTML, ou outro conteúdo, para ser processado...

Fonte: (BASHAM, SIERRA, BATES 2008), p. 17

Repare que o valor para o MIMÉ type refere-se aos valores listados

Uniform Resource Locator: é a URL, utilizada para a obtenção de recursos no servidor pelo navegador web.

Fonte: (BASHAM, SIERRA, BATES 2008), p. 20

Protocolo: Informa ao servidor qual protocolo de comunicação (neste caso o HTTP) que será usado.

Porta: Esta parte da URL é opcional. Um único servidor suporta várias portas. Uma aplicação é identificada por uma porta. Se você não especificar uma porta em sua URL, a porta 80 será o padrão e, coincidentemente, esta é a porta-padrão para os servidores web.

Recurso: O nome do conteúdo sendo solicitado. Pode ser uma página HTML, um servlet, uma imagem, PDF, música, vídeo ou qualquer outra coisa que o servidor queira disponibilizar. Se esta parte opcional for omitida, a maioria dos servidores irá procurar por index.html por padrão.

`http://www.wickedlysmart.com:80/beeradvice/select/beer1.html`

Servidor: O nome único do servidor físico pelo qual você está procurando. Este nome aponta para um único endereço IP. Os endereços IP são numéricos e assumem a forma "xxx.yyy.zzz.aaa". Você pode especificar um endereço IP aqui em vez de um nome, mas um nome é bem mais fácil de lembrar.

Caminho: O caminho para a localização, no servidor, do recurso que está sendo solicitado. Em virtude de a maioria dos servidores web mais novos rodar Unix, a sintaxe Unix ainda é usada para descrever as hierarquias de diretórios.

Verbos HTTP

- Cada requisição HTTP está atrelada a um verbo HTTP, ou método, no cabeçalho de requisição. São as letras maiúsculas no inicio do cabeçalho. Por exemplo,
- GET / HTTP/1.1
 - Significa que o método GET está sendo utilizado, enquanto
- DELETE /clientes/ana HTTP/1.1
 - significa que o método DELETE está sendo utilizado.
- Os verbos HTTP dizem ao servidor o que ele deve fazer com a informação identificada na URL.

Verbos HTTP

- A requisição pode por opção conter informações adicionais no corpo/body, que podem ser necessárias para executar a operação - por exemplo, informações que você quer guardar com o recurso.
- Como neste curso já criamos diversos formulários em HTML, estamos familiarizados com dois dos mais importantes verbos HTTP: GET e POST. Mas, existem muitos outros verbos HTTP disponíveis. Os mais importantes para criar uma API RESTful são GET, POST, PUT, e DELETE. Outros métodos estão disponíveis, como o HEAD e OPTIONS, mas a utilização destes é mais rara (se você quer saber mais sobre métodos HTTP, acesse a RFC 2616 em <http://www.ietf.org/rfc/rfc2616.txt>).

Verbos HTTP usados no REST

- Os verbos HTTP são associados então com os tradicionais CRUD para a criação da aplicação, conforme abaixo:
- **C**reate (insert) = POST
- **R**etrieve (select) = GET
- **U**pdate = PUT
- **D**elete = DELETE

JSON

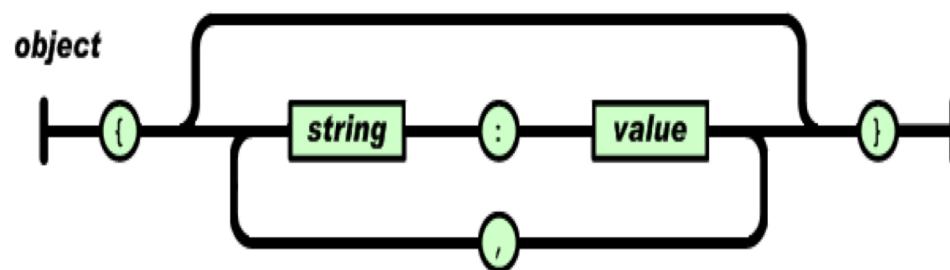
- JSON ([JavaScript Object Notation - Notação de Objetos JavaScript](#)) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar.
- Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição -Dezembro - 1999.
- JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras.
- Estas propriedades fazem com que JSON seja um formato ideal de troca de dados.

JSON está constituído em duas estruturas

- **Uma coleção de pares nome/valor.** Em várias linguagens, isto é caracterizado como um object, record, struct, dicionário, hash table, keyed list, ou arrays associativos.
- **Uma lista ordenada de valores.** Na maioria das linguagens, isto é caracterizado como um array, vetor, lista ou sequência.
- Estas são estruturas de dados universais. Virtualmente todas as linguagens de programação modernas as suportam, de uma forma ou de outra. É aceitável que um formato de troca de dados que seja independente de linguagem de programação se baseie nestas estruturas

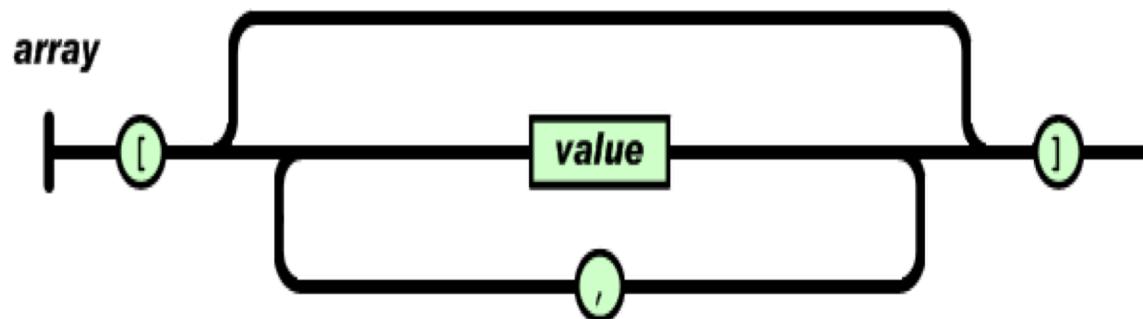
Objeto

- Um objeto é um conjunto desordenado de pares nome/valor. Um objeto começa com { (chave de abertura) e termina com } (chave de fechamento). Cada nome é seguido por : (dois pontos) e os pares nome/valor são seguidos por , (vírgula).



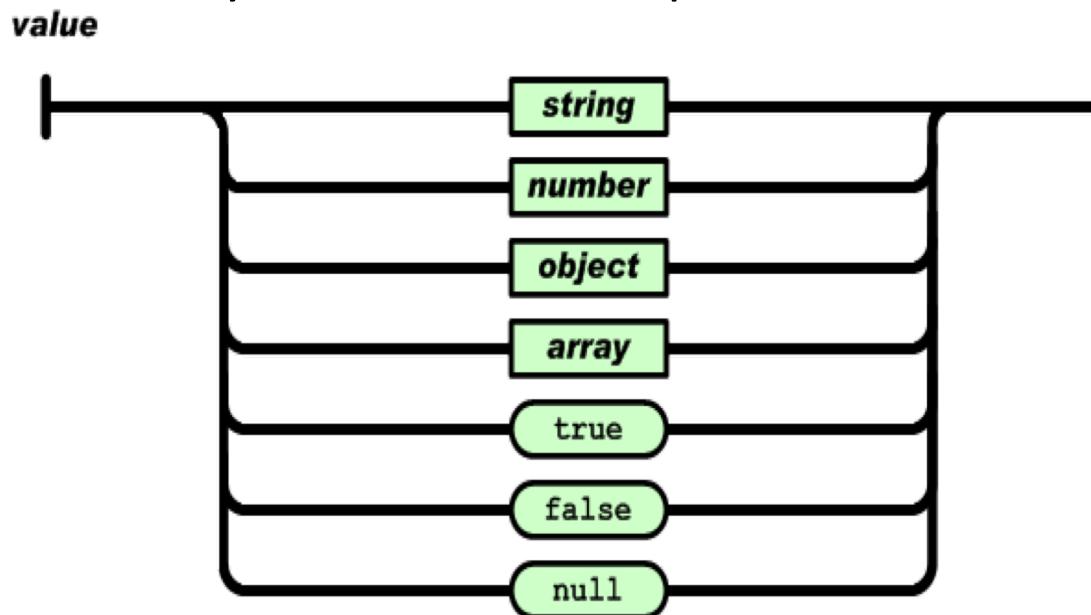
Array

- Um array é uma coleção de valores ordenados. O array começa com [(colchete de abertura) e termina com] (colchete de fechamento). Os valores são separados por , (vírgula).



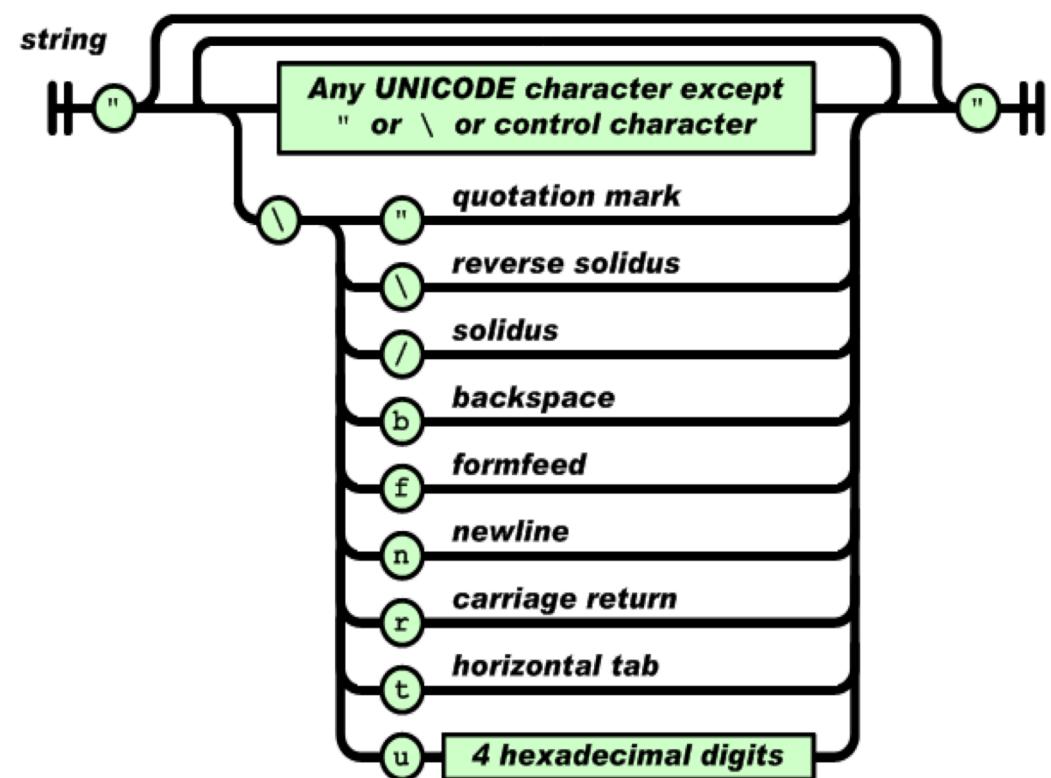
Valor

- Um valor (value, na imagem acima) pode ser uma cadeia de caracteres (string), ou um número, ou true ou false, ou null, ou um objeto ou um array. Estas estruturas podem estar aninhadas.



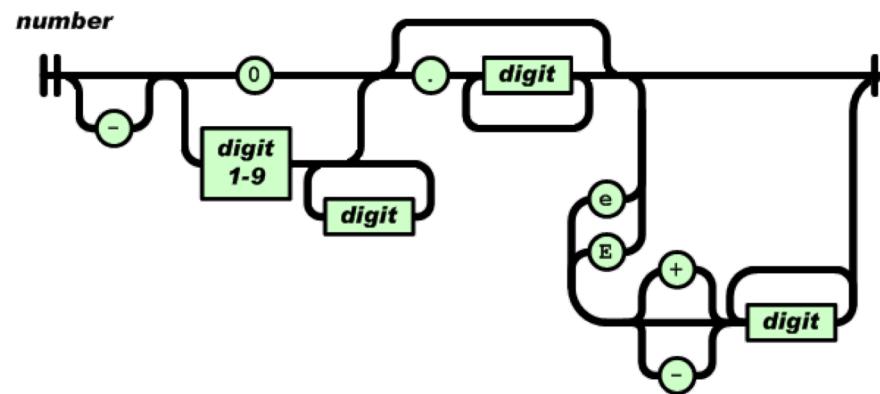
String

- Uma string é uma coleção de nenhum ou mais caracteres Unicode, envolvido entre aspas duplas usando barras invertidas como caractere de escape. Um caractere está representando como um simples caractere de string. Uma cadeia de caracteres é parecida com uma cadeia de caracteres em C ou Java.



Número

- Um número é similar a um número em C ou Java, exceto quando não se usa os números octais ou hexadecimais.



Exemplo de arquivo JSON

```
[  
  {  
    "id": 1,  
    "nome": "Capela USJT",  
    "latitude": -23.551641,  
    "longitude": -46.597417,  
    "imagem": null,  
    "cidade": {  
      "id": 1,  
      "nome": "São Paulo",  
      "estado": {  
        "id": "SP",  
        "nome": "São Paulo"  
      }  
    },  
    "tipo": {  
      "id": 1,  
      "nome": "pokestop"  
    }  
  },  
  {  
    "id": 2,  
    "nome": "Alberto M. Camargo",  
    "latitude": -23.552166,  
    "longitude": -46.597626,  
    "imagem": null,  
    "cidade": {  
      "id": 1,  
      "nome": "São Paulo",  
      "estado": {  
        "id": "SP",  
        "nome": "São Paulo"  
      }  
    },  
    "tipo": {  
      "id": 1,  
      "nome": "pokestop"  
    }  
  },  
  {  
    "id": 3,  
    "nome": "Estátua Fundador",  
    "latitude": -23.551622,  
    "longitude": -46.597847,  
    "imagem": null,  
    "cidade": {  
      "id": 1,  
      "nome": "São Paulo",  
      "estado": {  
        "id": "SP",  
        "nome": "São Paulo"  
      }  
    },  
    "tipo": {  
      "id": 1,  
      "nome": "pokestop"  
    }  
  }]
```

Exemplo de REST/JSON com Spring

Adicionar os jars do Jackson nas libs

- O Jackson faz o parse de objetos Java diretamente para o JSON
- Não é preciso configurar o spring-context para usá-lo, basta usar alguma anotações no código.



jackson-annotations-2.7.7.jar
jackson-core-2.7.7.jar
jackson-databind-2.7.7.jar

Anotações e Objetos para REST

- **@RestController** – cria um controller restful
- **@RequestMapping(method=RequestMethod.GET, value="rest/locais")** – a **@RequestMapping** sozinha atende todos os verbos HTTP. Com method você escolhe o método, o que é importantíssimo para REST
- **@ResponseBody** - indica que a resposta será feita direta na response do HTTP. Só isto já é o suficiente para que o Jackson converta o objeto em JSON.
- **@RequestBody** – indica que o objeto será lido diretamente da request. Isto faz com que o Jackson converta o objeto JSON em objeto Java.
- **@PathVariable("id")** – faz com que o ID digitado na URL seja reconhecido como um parâmetro, e não um recurso
- **ResponseEntity** – objeto que coloca na response um objeto JSON e o status HTTP

Exemplo de RestController do Pokemapa

```
package br.usjt.arqdes16.mapeamento.controller;

import java.io.IOException;
import java.util.List;
import javax.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import br.usjt.arqdes16.mapeamento.model.Local;
import br.usjt.arqdes16.mapeamento.service.LocalService;
```

```
@RestController                                Continuação da classe MapeamentoRest
public class MapeamentoRest {

    private LocalService ls;

    @Autowired
    public MapeamentoRest(LocalService ls) {
        this.ls = ls;
    }

    @RequestMapping(method=RequestMethod.GET, value="rest/locais")
    public @ResponseBody List<Local> listagem(String chave) {
        List<Local> lista = null;
        try{
            if(chave == null || chave.equals("")){
                lista = ls.listarLocais();
            } else {
                lista = ls.listarLocais(chave);
            }
        } catch(IOException e){
            e.printStackTrace();
        }
        return lista;
    }
}
```

Continuação da classe MapeamentoRest

```
@RequestMapping(method=RequestMethod.GET, value="rest/locais/{id}")
public @ResponseBody Local listaLocal(@PathVariable("id") Long id) {
    Local local = null, param;
    try{
        param = new Local();
        param.setId(id.intValue());
        local = ls.mostrar(param);
    } catch(IOException e){
        e.printStackTrace();
    }
    return local;
}
```

Continuação da classe MapeamentoRest

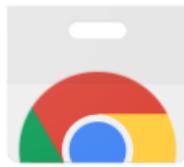
```
@Transactional  
@RequestMapping(method=RequestMethod.POST, value="rest/locais")  
public ResponseEntity<Local> criarLocal(@RequestBody Local local){  
    try{  
        ls.criar(local);  
        return new ResponseEntity<Local>(local, HttpStatus.OK);  
    } catch(IOException e){  
        e.printStackTrace();  
        return new ResponseEntity<Local>(local, HttpStatus.INTERNAL_SERVER_ERROR);  
    }  
}
```


Tratamento de Datas

- Os javabeans que possuirem atributos `java.util.Date` devem ter estes atributos anotados com:
- `@JsonFormat(pattern="dd-MM-yyyy")`
- Caso contrário as datas serão transformadas em números inteiros ao serem convertidas em JSON pelo Jackson.

Teste de Serviços REST

Postman: plugin do Chrome para acessar serviços RESTful



Web Store



Google Docs



YouTube



Gmail



Planilhas do Google

para instalar, vá ao Chrome Web Store e pesquise por postman

Instale esta versão.

Aplicativos

Mais resultados de Aplicativo



Postman
oferecido por www.getpostman.com

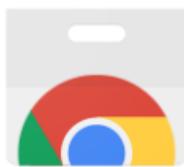
Supercharge your API workflow with Postman! Build, test, and document your APIs faster. More than a million developers already do....

+ USAR NO CHROME

Ferramentas do desenvolvedor

★★★★★ (7237)

Clique no Postman



Web Store



Google Docs



YouTube



Gmail

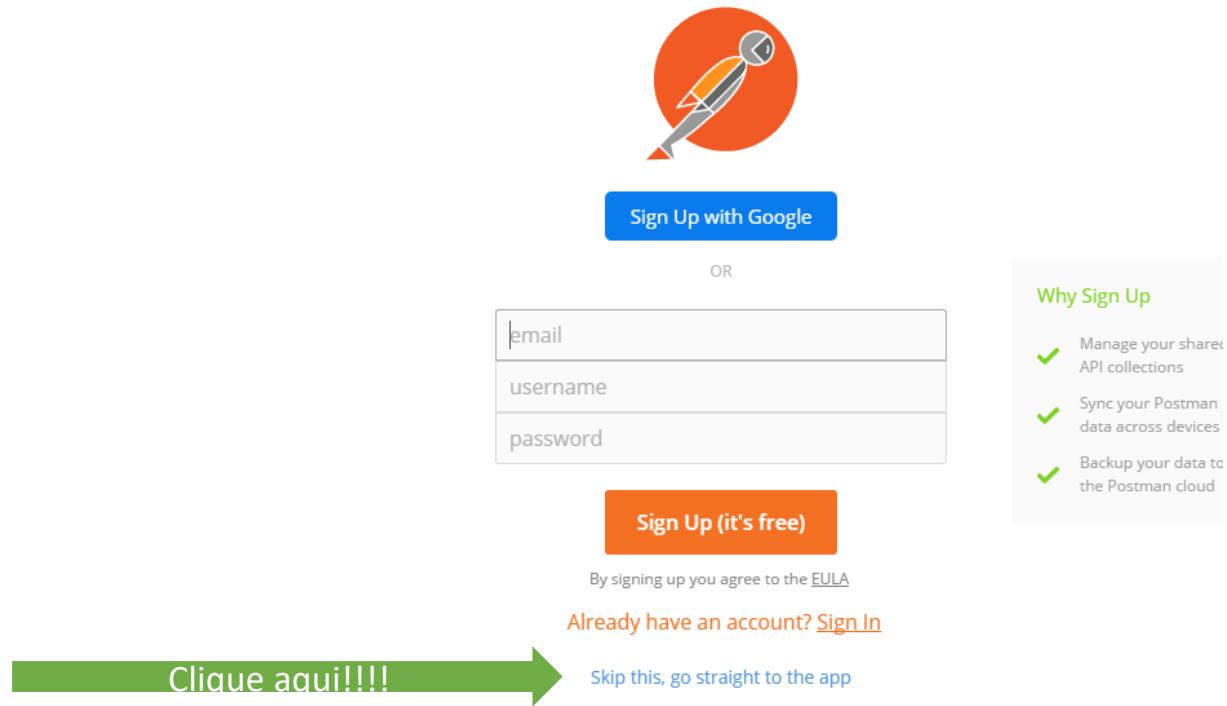


Planilhas do Google



Postman

Não precisa fazer login



The image shows the Postman sign-up interface. It features a large orange circular icon with a stylized pen and gear inside. Below it is a blue button labeled "Sign Up with Google". To the right of the button is the text "OR". Below "OR" are three input fields: "email", "username", and "password". Below these fields is an orange button labeled "Sign Up (it's free)". Underneath the button, the text "By signing up you agree to the [EULA](#)" is visible. To the right of the sign-up form is a light gray box titled "Why Sign Up" containing three bullet points: "Manage your shared API collections", "Sync your Postman data across devices", and "Backup your data to the Postman cloud". At the bottom left, a green bar contains the text "Clique aqui!!!!" followed by a green arrow pointing right. Next to the arrow is the text "skip this, go straight to the app".

Clique aqui!!!! → skip this, go straight to the app

No Environment

GET http://localhost:8080/pokemap4/rest/locais

Params

Send Save

Code

Type: No Auth

Authorization Headers Body Pre-request Script Tests

Body Cookies Headers (4) Tests

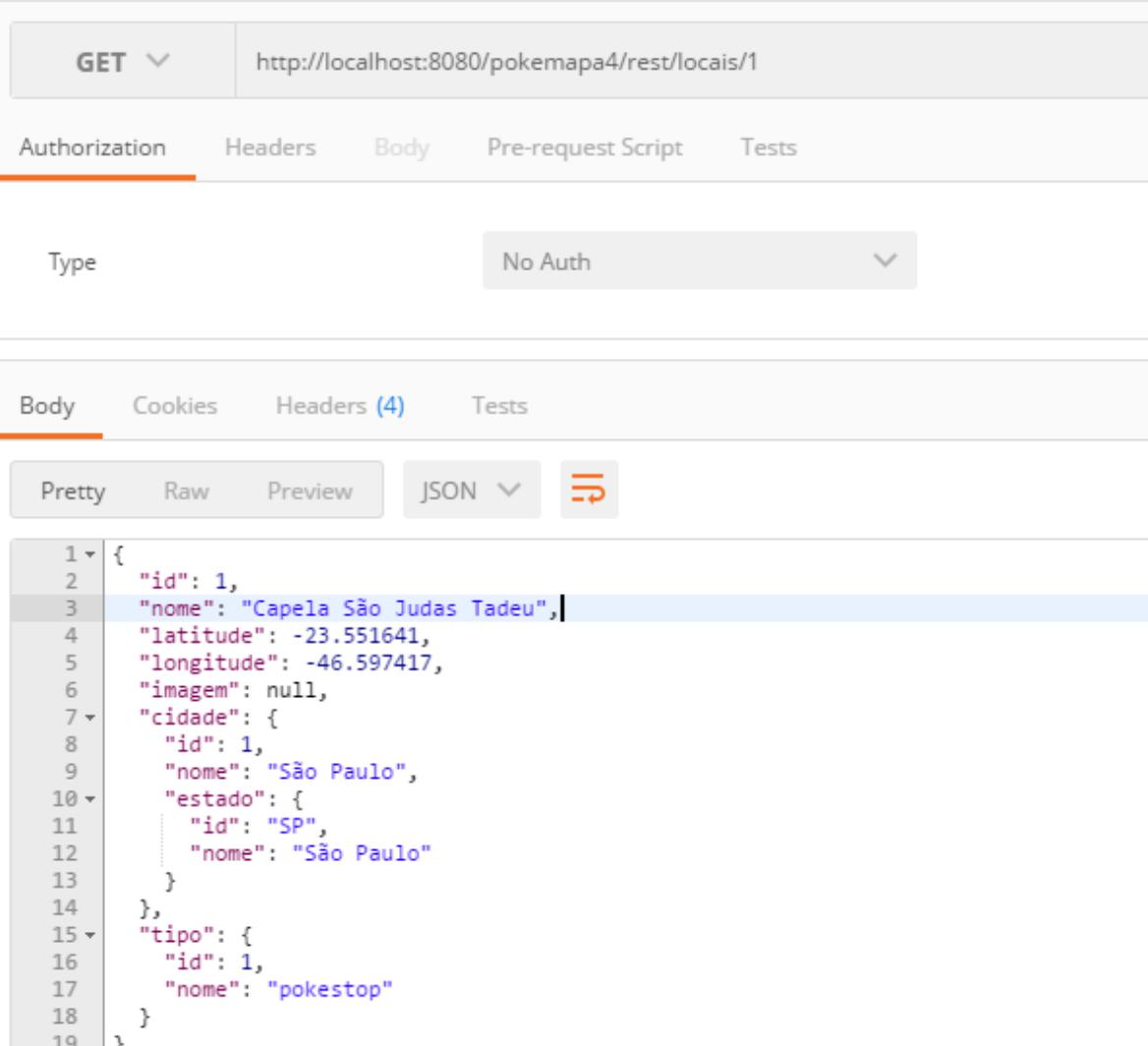
Pretty Raw Preview JSON

```
1 {  
2   "id": 1,  
3   "nome": "Capela São Judas Tadeu",  
4   "latitude": -23.551641,  
5   "longitude": -46.597417,  
6   "imagem": null,  
7   "cidade": {  
8     "id": 1,  
9     "nome": "São Paulo",  
10    "estado": {  
11      "id": "SP",  
12      "nome": "São Paulo"  
13    }  
14  },  
15  "tipo": {  
16    "id": 1,  
17    "nome": "pokestop"  
18  }  
19 }
```

Status: 200 OK Time: 197 ms

Preencha a url, escolha o método e clique em send. Neste caso estamos pesquisando todos

Pesquisa pelo código do local



The screenshot shows the Postman application interface. At the top, there is a header bar with a 'GET' dropdown and a URL field containing 'http://localhost:8080/pokemap4/rest/locais/1'. Below the header, there are tabs for 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Tests'. The 'Authorization' tab is currently selected, showing a dropdown menu with 'No Auth' selected. In the main body area, there are tabs for 'Body', 'Cookies', 'Headers (4)', and 'Tests'. The 'Body' tab is selected, displaying a JSON response. The JSON response is as follows:

```
1 | {
2 |   "id": 1,
3 |   "nome": "Capela São Judas Tadeu",
4 |   "latitude": -23.551641,
5 |   "longitude": -46.597417,
6 |   "imagem": null,
7 |   "cidade": {
8 |     "id": 1,
9 |     "nome": "São Paulo",
10 |     "estado": {
11 |       "id": "SP",
12 |       "nome": "São Paulo"
13 |     }
14 |   },
15 |   "tipo": {
16 |     "id": 1,
17 |     "nome": "pokestop"
18 |   }
19 | }
```

POST <http://localhost:8080/pokemapa4/rest/locais>

Authorization Headers (1) Body **Pre-request Script** Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {  
2   "nome": "Candangos",  
3   "latitude": -15.800533,  
4   "longitude": -47.860951,  
5   "imagem": null,  
6   "cidade": {  
7     "id": 3,  
8     "nome": "Brasília",  
9     "estado": {  
10       "id": "DF",  
11       "nome": "Distrito Federal"  
12     }  
13   },  
14   "tipo": {  
15     "id": 2,  
16     "nome": "ginásio"  
17   }  
18 }
```

Status: 200 OK Time: 1928 ms

Body Cookies Headers (4) Tests

Pretty Raw Preview JSON

```
1 {  
2   "id": 5,  
3   "nome": "Candangos",  
4   "latitude": -15.800533,  
5   "longitude": -47.860951,  
6   "imagem": null,  
7   "cidade": {  
8     "id": 3,  
9     "nome": "Brasília",  
10    "estado": {  
11      "id": "DF",  
12      "nome": "Distrito Federal"  
13    }  
14  },  
15  "tipo": {  
16    "id": 2,  
17    "nome": "ginásio"  
18  }  
19 }
```

Retorna o objeto
com o id e o
status http

Inclusão: passe
o objeto todo,
menos o id!

Bibliografia

- BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. Use a cabeça!: Servlets & JSP. 2. ed. Rio de Janeiro: Alta Books, 2008-2010. xxxii, 879 p. ISBN 9788576082941 (broch.)
- Introdução ao REST WS, disponível em <http://www.devmedia.com.br/introducao-ao-rest-ws/26964>. Acessado em 12/05/2015.
- Introdução ao JSON, disponível em <http://www.json.org/json-pt.html>. Acessado em 12/05/2015.