

Programação Para Internet

**Aula: Java Web Service -
REST**



Prof. Anderson Augusto Bosing

O que é REST?

Contexto?

O que é?

Para que serve?

Como?



Java Web Service - REST

Era em um época no qual implementávamos web services usando estilo SOAP -

[https://pt.wikipedia.org/wiki/SOAP.](https://pt.wikipedia.org/wiki/SOAP)

Nesse momento, REST foi anunciado como como um estilo de web services concorrente ao SOAP, no qual apresentava melhorias, leveza, facilidades e maior escalabilidade.

Java Web Service - REST

E assim, o REST foi ao poucos sendo adicionadas plataformas e passou a ser utilizado com mais uma opção na criação de web services.

A partir de então, os desenvolvedores possuíam duas diferentes opções para fazer web services SOAP ou REST.

Java Web Service - REST

“Representational State Transfer” (REST) foi a sigla apresentada pela tese de doutorado de Roy Fielding, autor do protocolo HTTP, que defende a utilização do protocolo HTTP puro como web service sem a necessidade de especificações de protocolos adicionais como o SOAP e etc.

No REST, Roy estabeleceu um conjunto de princípios que definem como os recursos padrões de comunicação existentes no HTTP devem ser usados para ser utilizado como solução de web services.



Java Web Service - REST

Assim, REST não criou e nem inventou nada de novo, simplesmente passou a usar o antigo e velho conhecido protocolo HTTP como solução final para criação de web services para oferecer integração 100% interoperável.

Java Web Service - REST

REST é:

Uma técnica de engenharia de engenharia de software que faz uso do protocolo HTTP como um estilo arquitetural utilizado para comunicação (troca de informações) entre sistemas distribuídos que lidam simultaneamente com diversos formatos textos, imagens, vídeos, etc.



Java Web Service - REST

RESTFul é nome dado para as soluções que seguem os princípios de REST como solução de web services.

Java Web Service - REST

Para poder criar web services REST com eficiência é necessário conhecer pelo menos básico do protocolo

HTTP:

Wiki:

https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Especificação:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>



Java Web Service - REST

Para usar HTTP como web services, é necessário seguir os cinco princípios fundamentais:

1. Dê a todas as coisas um identificador [URL].
2. Utilize protocolo HTTP para definir as ações e suas respostas[HTTP].
3. Recursos com múltiplas representações [MIME].
4. Comunique sem estado [Stateless].
5. Vincule as coisas [links].

Java Web Service - REST

1. Dê a todas as coisas um identificador:

Todos os serviços oferecido no seu web services rest deverá ter um identificador unico na web. Use URIs para identificar tudo o que precisar ser identificado, especifique todos os recursos de "alto nível" que seu aplicativo oferece, se eles representam itens individuais, conjuntos de itens, objetos virtuais e físicos, ou resultados de computação.

Java Web Service - REST

1. Dê a todas as coisas um identificador:

Assim, uma solução cliente pode interagir com um recurso conhecendo apenas seu identificador (URL) não necessitando conhecer a infraestrutura que a faz funcionar.

Isso é: baixo acoplamento, independência de plataforma e nenhuma necessidade de protocolos adicionais.



Java Web Service - REST

1. Dê a todas as coisas um identificador:

Exemplos:

<http://vendas.com/cliente/1234>

<http://vendas.com/vendas/2007/10/776654>

<http://vendas.com/produtos/4554>

<http://vendas.com/boleto/5248>



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Use os métodos do protocolo HTTP para especificar quais as ações serão efetuadas sobre os recursos disponibilizado nas URI's.

Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Mapeando para um CRUD básico, ficaria assim os quatro mais utilizados:

- ▶ Create - POST
- ▶ Read - GET
- ▶ Update - PUT
- ▶ Delete - DELETE

Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Segue exemplos crud:

▶ POST `http://vendas.com/cliente` = gravar dados do cliente que será enviado dentro do corpo do protocolo HTTP.

▶ GET `http://vendas.com/cliente/1234` = obter dados do cliente 1234

▶ PUT `http://vendas.com/cliente/1234` = atualizar dados do cliente que será enviado dentro do corpo do protocolo HTTP.

▶ DELETE `http://vendas.com/cliente/1234` = deletar do sistema o cliente 1234



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Use status HTTP que define a resposta ao cliente daquela determinada requisição:

- ▶ HTTP 1xx – Informativo.
- ▶ HTTP 2xx – Sucesso.
- ▶ HTTP 3xx – Redirecionamento.
- ▶ HTTP 4XX – Erro do cliente
- ▶ HTTP 5XX – Erro do servidor.

Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

▶ **SUCESSO 200 – OK** – indica que o pedido foi processado com sucesso. Usado com GET para retornar o(s) recurso(s) requerido(s).

▶ **SUCESSO 201 – CRIADO** – indica que o(s) recurso(s) do pedido foram criadas com sucesso. Deve retornar no header da resposta a URI para o(s) recurso(s) criado(s). Usado com POST E PUT.

Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

- ▶ **SUCESSO 202 – APROVADO** – indica que o pedido foi aprovado mas não processado. Utilizado para processamento assíncrono.
- ▶ **SUCESSO 204 – SEM CONTEÚDO** – indica que o pedido foi processado com sucesso. Usado com UPDATE e DELETE sem resposta (não tem BODY).
- ▶ **ERRO CLIENTE 400 – PEDIDO INVÁLIDO** – indica que o cliente enviou o pedido com sintaxe errada.

Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

- ▶ **ERRO CLIENTE 401 – NÃO AUTORIZADO** – indica que o pedido foi recurso por requerer credencial não existente ou inválidas.
- ▶ **ERRO CLIENTE 404 – NÃO ENCONTRADO** – indica que o servidor não encontrou nada que corresponda á URI do pedido.
- ▶ **ERRO SERVIDOR 500 – ERRO INTERNO DO SERVIDOR** – o servidor encontrou uma condição inesperado que o impediu que cumprir a solicitação.

Java Web Service - REST

3. Recursos com múltiplas representações:

Use o cabeçalho padrão de MIME do protocolo HTTP para especificar pelo cliente qual será o tipo do resultado esperado por ele na resposta das URI's, fazendo o próprio cliente decidir qual é o tipo do resultado que ele espera receber.

Java Web Service - REST

3. Recursos com múltiplas representações:

Use o cabeçalho padrão de MIME do protocolo HTTP para especificar pelo servidor qual será o tipo do resultado da resposta disponibilizado nas URI's fazendo com que um mesmo recurso possa ter dinamicamente múltiplos representações diferentes.

Java Web Service - REST

3. Recursos com múltiplas representações:

Exemplos: Resposta de pedido do cliente em txt:

HTTP-Version: HTTP/1.0 200 OK

Content-Length: 3012

Content-Type: text/txt

<body>

Ricardo, 25 anos de idade, engenheiro.

</body>



Java Web Service - REST

4. Comunique sem estado:

Crie seu web services totalmente STATELESS, sem armazenar estado do cliente no servidor. Isto permite alguns benefícios como:

- ▶ **Visibilidade:** não é necessário verificar a requisição para determinar sua natureza;
- ▶ **Fiabilidade:** a tarefa de recuperação a falhas parciais se torna fácil;
- ▶ **Escalabilidade:** não necessitando armazenar estados entre as requisições, o servidor pode rapidamente liberar recursos facilitando seu gerenciamento e ter escalabilidade horizontal fácil e rápida.

Java Web Service - REST

5. Vincule as coisas:

Sempre que necessário e quando houver aplicabilidade, use links para referenciar outros recursos que possam ser identificados dinamicamente, permitindo que a hipermídia funcione como máquina de estado da solução, a partir da própria aplicação, sem interferência humana. Conceito complexo chamado de "HATEOAS":

Java Web Service - REST

“Na teoria eu sei que tenho que tentar seguir, mas na prática eles realmente funcionam?”



Princípios - REST

Princípios são ideais e valores almejados, mas que na prática e mundo real, em certos casos podem não ser aplicáveis.

Sendo assim, tente fazer sua solução web service rest o mais RESTFUL possível, seguindo esses princípios.

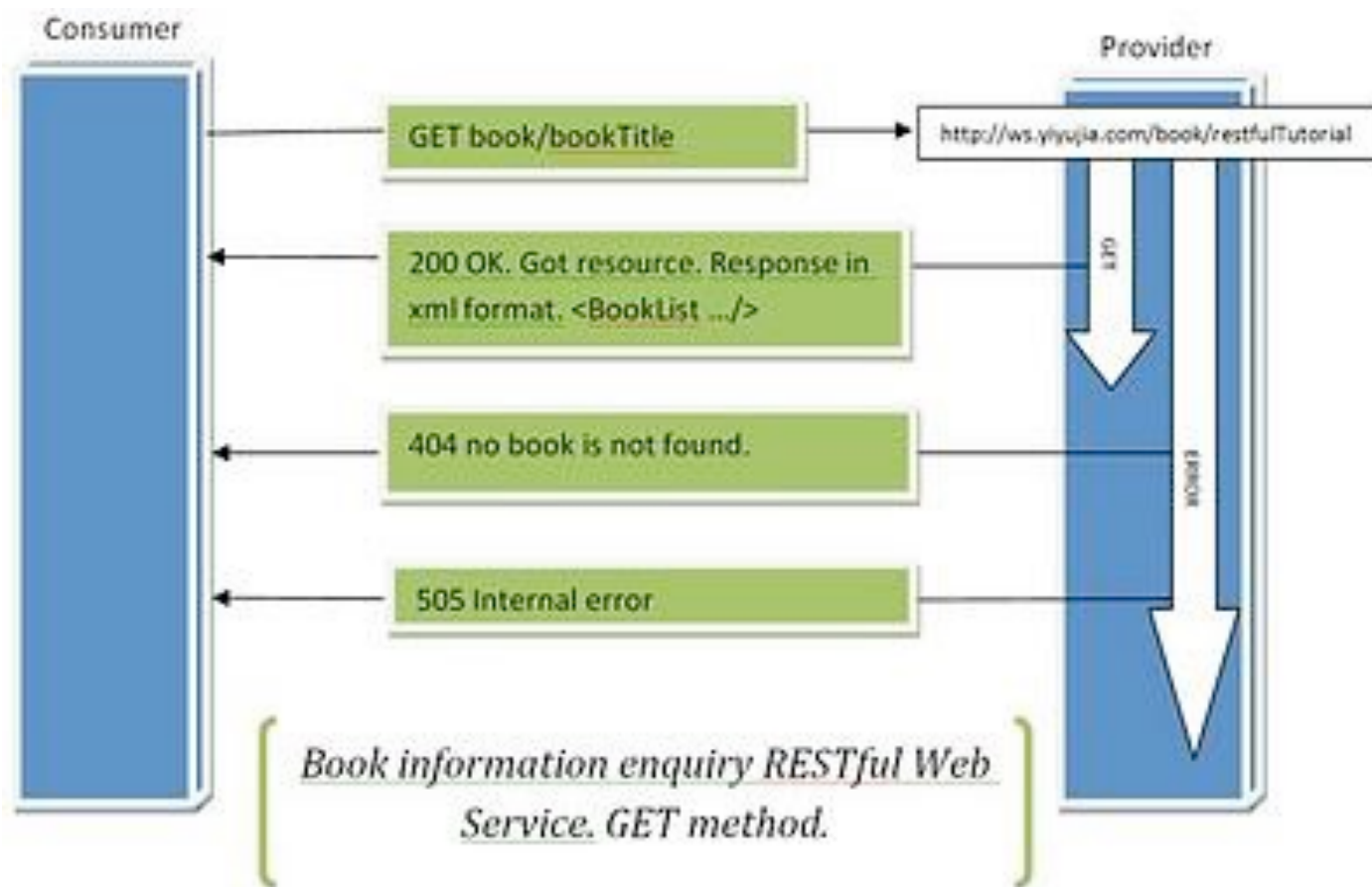
Se começar um projeto novo, do zero, siga eles!

Princípios - REST

Dica do Professor:

- ▶ Na prática, conseguimos usar os 3 primeiros princípios que são fundamentais e básicos para fazer um rest real.
- ▶ No caso do HATEOAS, somente em casos que existe aplicabilidade e ganho real de uso. Caso contrário, nem usamos isso. Eu não nunca precisei usar.

Arquitetura - REST



Princípios - REST

Workflow de execução:

1. A solução servidora disponibiliza a lista de recursos através de URI's e métodos HTTP.
2. A solução consumidora cria e envia uma requisição HTTP para o web service REST, usando uma URL e um específico método HTTP.

Princípios - REST

Workflow de execução:

3. A solução cliente envia informações via parâmetros HTTP dentro do protocolo ou na própria URL destino.
4. A solução servidora descobre o que o cliente quer pela URL e método HTTP. O processamento é feito retornando algum código de resposta HTTP válido e ou algum recurso dentro da resposta HTTP.

Princípios - REST

Workflow de execução:

5. A solução cliente consumidora descobre qual foi a resposta (sucesso, erro, redirecionamento, etc) pelo código da resposta HTTP, juntamente do conteúdo vindo do processamento dentro da resposta do protocolo HTTP.

Princípios - REST

URIs e métodos HTTP do processo de leilão

URI	Método	Formato	Efeito
/item/{id}	GET	Item	Busca um item.
	PUT	Item	Atualiza um item.
/item/{id}/ofertas	GET	Coleção de ofertas	Busca ofertas feitas sobre um item.
	POST	Oferta	Adiciona oferta a um item.
/oferta/{id}	GET	Oferta	Busca uma oferta.
	PUT	Oferta	Atualiza uma oferta.
	DELETE	-	Remove uma oferta.
/usuario	POST	Usuario	Cadastra um usuário.
/usuario/{id}	GET	Usuario	Busca um usuário.
	PUT	Usuario	Atualiza um usuário.
/usuario/{id}/avaliacoes	GET	Coleção de avaliações	Busca as avaliações recebidas por um usuário.
/usuario/{id}/itens	GET	Coleção de itens	Busca os itens anunciados por um determinado usuário.
	POST	Item	Usuário coloca novo item à venda.
/avaliacao/{id}	GET	Avaliação	Busca uma determinada avaliação.
/avaliacao/de/{id}/para/{id}	POST	Avaliação	Realização da avaliação de um usuário sobre outro.
/services	GET	Coleção de URIs	Consulta URIs e métodos HTTP disponíveis para acesso.

Protocolos de comunicação

Requisições HTTP

Requisições HTTP são mensagens enviadas pelo cliente para iniciar uma ação no servidor. Suas linhas iniciais contêm três elementos:

1) Um método HTTP, um verbo (como GET, PUT ou POST) ou um nome (como HEAD ou OPTIONS), que descrevem a ação a ser executada. Por exemplo, GET indica que um recurso deve ser obtido ou POST significa que dados são inseridos no servidor (criando ou modificando um recurso, ou gerando um documento temporário para mandar de volta).

Protocolos de comunicação

Requisições HTTP

2) O alvo da requisição, normalmente um URL, ou o caminho absoluto do protocolo, porta e domínio são em geral caracterizados pelo contexto da requisição. O formato deste alvo varia conforme o método HTTP.

Sintaxe geral de uma URL:

<protocolo>://<servidor>:<porta>/<caminho>/<recurso>

- A porta é opcional para serviços em portas default
- Caminho e recurso podem ser omitidos (URLs parciais)
- URLs podem conter dados depois do nome do recurso

Protocolos de comunicação

Requisições HTTP

Exemplos de URLs:

<http://java.sun.com/docs/servlets/servlets.html>

<http://java.sun.com/docs/servlets/>

<http://java.sun.com/cgi-bin/reverse?string=fred>

<http://localhost:8080/fred/servlets/ListaServlet?tipo=superior&curso=334>



Protocolos de comunicação

Cabeçalhos HTTP

Cabeçalhos HTTP de uma requisição seguem a mesma estrutura básica de um cabeçalho HTTP: uma cadeia de caracteres insensível à caixa seguida de dois pontos (':') e um valor cuja estrutura depende do cabeçalho. O cabeçalho inteiro, incluindo o valor, consiste em uma única linha, que pode ser bem grande. Há numerosos cabeçalhos de requisição disponíveis.

Protocolos de comunicação

Cabeçalhos HTTP

POST / HTTP/1.1

Host: localhost:8000

User-Agent: Mozilla/5.0 (Macintosh;...)... Firefox/51.0

Accept: text/html,application/xhtml+xml,..., */*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: multipart/form-data; boundary=-12656974

Content-Length: 345

Request headers

General headers

Entity headers

-12656974

(more data)

Protocolos de comunicação

Corpo HTTP

Corpo A parte final da requisição é o corpo. Nem todas as requisições tem um: as que pegam recursos, como GET, HEAD, DELETE, ou OPTIONS, usualmente não precisam de um. Algumas requisições enviam dados ao servidor a fim de atualizá-lo: é o caso frequente de requisições POST (contendo dados de formulário HTML)

Protocolos de comunicação

Respostas HTTP

A linha inicial de uma resposta HTTP, chamada de linha de status, contém a seguinte informação: A versão do protocolo, normalmente HTTP/1.1. Um código de status, indicando o sucesso ou falha da requisição. Códigos de status comuns são 200, 404, ou 302 Um texto de status. Uma descrição textual breve, puramente informativa, do código de status a fim de auxiliar o entendimento da mensagem HTTP por humanos. Uma linha de status típica se parece com: HTTP/1.1 404 Not Found.

Protocolos de comunicação

Verbos HTTP

A ideia geral é a seguinte: seu serviço vai prover uma url base e os verbos HTTP vão indicar qual ação está sendo requisitada pelo consumidor do serviço.

Verbos HTTP

DELETE

O método de requisição HTTP DELETE remove o recurso especificado.

Exemplo:

DELETE maxrest/rest/mbo/asset/1234 HTTP/1.1



Verbos HTTP

GET

O método HTTP GET solicita uma representação do recurso especificado. Solicitações usando GET só devem recuperar dados.

Exemplo:

GET maxrest/rest/mbo/asset/1234 HTTP/1.1



Verbos HTTP

POST

O método HTTP POST envia dados ao servidor. O tipo do corpo da solicitação é indicado pelo cabeçalho Content-Type.

Exemplo:

POST / HTTP/1.1

Host: foo.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 13

say=Hi&to=Mom



Verbos HTTP

PUT

O método de requisição HTTP PUT cria um novo recurso ou substitui uma representação do recurso de destino com os novos dados.

Exemplo:

PUT /new.html HTTP/1.1

Host: example.com

Content-type: text/html

Content-length: 16

<p>New File</p>



HTTP Response Status Code 100 Continue

O Status HTTP 100 Continue indica que até o momento tudo está OK e que o cliente pode continuar com a requisição ou ignorar caso já tenha terminado.



HTTP Response Status Code

200 OK

O código HTTP 200 OK é a resposta de status de sucesso que indica que a requisição foi bem sucedida. Uma resposta 200 é cacheável por padrão.

O significado de sucesso depende do método de requisição HTTP:

GET: O recurso foi carregado e transmitido no corpo da mensagem.

POST: O recurso descrevendo o resultado da ação é transmitido no corpo da mensagem.



HTTP Response Status Code

200 OK

O resultado de sucesso de um PUT ou DELETE geralmente não são 200 OK, e sim 204 No Content (ou 201 Created quando o recurso é carregado pela primeira vez).

HTTP Response Status Code 201 Created

O status HTTP "201 Created" é utilizado como resposta de sucesso, indica que a requisição foi bem sucedida e que um novo recurso foi criado. Este novo recurso é efetivamente criado antes do retorno da resposta e o novo recurso é enviado no corpo da mensagem (pode vir na URL ou na header Location).

Comumente, este status é utilizado em requisições do tipo POST.



HTTP Response Status Code 400 Bad Request

O código de status de resposta HTTP 400 Bad Request indica que o servidor não pode ou não irá processar a requisição devido a alguma coisa que foi entendida como um erro do cliente (por exemplo, sintaxe de requisição mal formada, enquadramento de mensagem de requisição inválida ou requisição de roteamento enganosa).

HTTP Response Status Code 401 Unauthorized

O código de resposta de status de erro do cliente HTTP 401 Unauthorized indica que a solicitação não foi aplicada porque não possui credenciais de autenticação válidas para o recurso de destino.



HTTP Response Status Code 403 Forbidden

O código de resposta de status de erro do cliente HTTP 403 Forbidden indica que o servidor entendeu o pedido, mas se recusa a autorizá-lo.

Esse status é semelhante ao 401 , mas neste caso, a re-autenticação não fará diferença. O acesso é permanentemente proibido e vinculado à lógica da aplicação (como uma senha incorreta).

HTTP Response Status Code 404 Not Found

A resposta de erro 404 Not Found indica que o servidor não conseguiu encontrar o recurso solicitado. Normalmente, links que levam para uma página 404 estão quebrados ou desativados.

Um código 404 não indica se o recurso está indisponível temporariamente ou se o recurso foi permanentemente removido.



HTTP Response Status Code 405 Method Not Allowed

Este status de resposta indica que o verbo HTTP utilizado não é suportado, por exemplo: a requisição ocorre por meio de um get, porém o único método disponível é o post. Curiosidade: Existem um método chamado OPTIONS que retorna todos os verbos suportados naquela requisição. Obs: ele também pode não ser permitido

HTTP Response Status Code 500 Internal Server Error

Quando o servidor retorna um código de erro (HTTP) 500, indica que encontrou uma condição inesperada e que o impediu de atender à solicitação.

Essa resposta de erro é uma resposta genérica "abrangente". Às vezes, os arquivos log de servidores podem responder com um status code 500 acompanhado de mais detalhes sobre o request para evitar que no futuro erros desse tipo possam voltar a acontecer.

HTTP Response Status Code 503 Service Unavailable

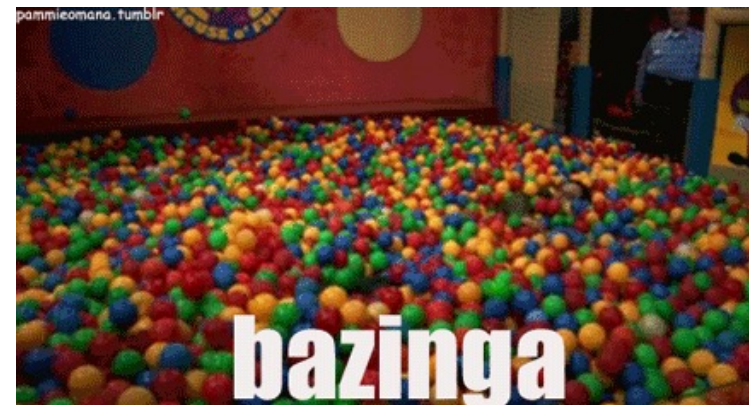
O código de resposta de erro de servidor 503 Service Unavailable do HTTP indica que o servidor não está pronto para lidar com a requisição.

Causas comuns são um servidor que está em manutenção ou sobrecarregado. Esta resposta deve ser usada para condições temporárias



HTTP Response Status Code 418 I'm a teapot

O código de erro HTTP para o cliente 418 I'm a teapot indica que o servidor se recusa a preparar café por ser um bule de chá. Este erro é uma referência ao Hyper Text Coffee Pot Control Protocol, que foi uma piada de 1º de abril de 1998.



JSON x XML

- Ambos são formatos de dados para recebimento e envio de dados com servidores web.
- Os dois modelos representam informações no formato texto.
- Ambos possuem natureza auto-descritiva (ou seja, basta “bater o olho” em um arquivo JSON ou em um arquivo XML para entender o seu significado). etc.
- Ambos são capazes de representar informação complexa, difícil de representar no formato tabular. Alguns exemplos: objetos compostos (objetos dentro de objetos), relações de hierarquia, atributos multivalorados, arrays, dados ausentes, etc.

JSON x XML

- Ambos podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627.
- Ambos são independentes de linguagem. Dados representados em XML e JSON podem ser acessados por qualquer linguagem de programação, através de API's específicas (API javascript por exemplo).

JSON x XML

```
{ "employees": [  
  { "firstName": "John", "lastName": "Doe" },  
  { "firstName": "Anna", "lastName": "Smith" },  
  { "firstName": "Peter", "lastName": "Jones" }  
]}
```

```
<employees>  
  <employee>  
    <firstName>John</firstName> <lastName>Doe</lastName>  
  </employee>  
  <employee>  
    <firstName>Anna</firstName> <lastName>Smith</lastName>  
  </employee>  
  <employee>  
    <firstName>Peter</firstName> <lastName>Jones</lastName>  
  </employee>  
</employees>
```

Criando nossa primeira API com Java – Principais Tecnologias Envolvidas

- ✓ JAX-RS é uma especificação que permite criar RESTful Web services (análoga a JAX-WS para SOAP).
- ✓ Jersey é a principal implementação da especificação JAX-RS.
- ✓ Wildfly é o servidor Web utilizado para executar as aplicações Java.



Instalação das ferramentas

IntelliJ

JDK

Wildfly

Postman (Documentação Collection, Requests)

Console do Navegador – Chrome/Firefox/Edge/Opera.

GIT



Instalação das ferramentas

Wildfly 31.0.1

Para adicionar um usuário admin ao wildfly precisamos executar o seguinte processo:

Acesse o seguinte path onde a instalação do wildfly foi descompactada: `~/wildfly-26.1.3.Final/bin`

Execute o script add-user.

Linux/Mac

`./add-user.sh`

Windows

`add-user.bat`



Instalação das ferramentas

```
bin — andersonbosing@MacBook-Air-de-Anderson — ..1.3.Final/bin — -zsh — 150x37
Last login: Sun Feb 26 08:45:46 on console
~/wildfly-26.1.3.Final/bin
> ./add-user.sh
What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): a
Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : unipar
Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.
- The password should be different from the username
- The password should not be one of the following restricted values {root, admin, administrator}
- The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password :
WFLYDM0102: Password should have at least 1 non-alphanumeric symbol.
Are you sure you want to use the password entered yes/no? yes
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]:
About to add user 'unipar' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'unipar' to file '/Users/andersonbosing/wildfly-26.1.3.Final/standalone/configuration/mgmt-users.properties'
Added user 'unipar' to file '/Users/andersonbosing/wildfly-26.1.3.Final/domain/configuration/mgmt-users.properties'
Added user 'unipar' with groups to file '/Users/andersonbosing/wildfly-26.1.3.Final/standalone/configuration/mgmt-groups.properties'
Added user 'unipar' with groups to file '/Users/andersonbosing/wildfly-26.1.3.Final/domain/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server Jakarta Enterprise Beans calls.
yes/no? yes
To represent the user add the following to the server-identities definition <secret value="dW5pcGFyMTIz" />
~/wildfly-26.1.3.Final/bin
>
```



Por que o Standalone ?

https://docs.wildfly.org/19.1/Getting_Started_Guide.html#wildfly-10-configurations



Criando conexão com o Banco de Dados Postgres no Wildfly.

Add Datasource

Choose Template

Attributes

JDBC Driver

Connection

Test Connection

Review

1

2

3

4

5

6

Choose one of the predefined templates to quickly add a datasource or choose "Custom" to specify your own settings.

☐ Custom

☐ H2

☒ PostgreSQL

☐ MySQL

☐ MariaDB

☐ Oracle

☐ Microsoft SQLServer

☐ IBM DB2

☐ Sybase

Cancel

< Back

Next >

Add Datasource

Choose Template

Attributes

JDBC Driver

Connection

Test Connection

Review

1

2

3

4

5

6

Help

Name

Petdogui

JNDI Name *

java:/Petdogui

Required fields are marked with *

Cancel

< Back

Next >



Add Datasource

×

Choose Template

Attributes

JDBC Driver

Connection

Test Connection

Review

1

2

3

4

5

6

Help

Driver Name *

postgresql-42.5.4.jar

Driver Module Name

org.postgresql

Driver Class Name

org.postgresql.Driver

Required fields are marked with *

Cancel

< Back

Next >



Add Datasource

Choose Template

Attributes

JDBC Driver

Connection

Test Connection

Review

1

2

3

4

5

6

Help

Connection URL

jdbc:postgresql://localhost:5432/app

User Name

unipar

Password

unipar123

Security Domain

Credential Reference Store

Credential Reference Alias

Credential Reference Clear T...

Credential Reference Type

Cancel

< Back

Next >



Add Datasource

Choose Template

Attributes

JDBC Driver

Connection

Test Connection

Review

1


2

3

4

5

6



Test Connection Successful

Successfully tested connection for datasource **PetdoguiDS**.

Cancel

< Back

Next >



JAX-RS Annotations

Annotation

[@GET](#)

[@Produces](#)

[@Path](#)

[@PathParam](#)

[@QueryParam](#)

[@POST](#)

[@Consumes](#)

[@PUT](#)

[@DELETE](#)

[@ApplicationPath](#)

Pacote/Import

import javax.ws.rs.GET;

import javax.ws.rs.Produces;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.QueryParam;

import javax.ws.rs.POST;

import javax.ws.rs.Consumes;

import javax.ws.rs.PUT;

import javax.ws.rs.DELETE;

Import javax.ws.rs.ApplicationPath



Anotações – JAX-RS

@Path("/cliente") - Transforma uma classe Java em um recurso REST disponível via HTTP.

Devem ser aplicadas na declaração das classes criando um path relativo de URL padrão de acesso.

Anotações – JAX-RS

@Path("/abertos")

Quando aplicados nos métodos Java (expostos como métodos HTTP) definem um path específico de execução, resultando na concatenação como URL de acesso.

Modelo = path relativo + path recurso + path método.

Exe: <http://www.sistema.com/cliente/abertos>

Anotações – JAX-RS

`@Path("/abertos/{nomeusuario}")`

Podem ser usados para expressar parâmetros variáveis com “{” que serão preenchidas na URL em tempo de execução, nas invocações do serviços.

Estas variáveis são automaticamente disponíveis no métodos de serviços via DI pelo JAX-RS usando a anotação `@PathParam`.

Anotações – JAX-RS

@GET, @POST, @PUT, @DELETE

Mapeia os métodos da classe Java para responder aos correspondentes métodos do protocolo HTTP no qual serão automaticamente invocados pelo JAX-RS.

Devem ser aplicadas somente na declaração dos métodos da classe recurso.

Anotações – JAX-RS

@QueryParam

Utilizado para extrair informações enviados dentro da URL de uma requisição HTTP.

@DefaultValue

Utilizado para definir valores padrões para informações que não foram informados dentro da URL de uma requisição HTTP.

Anotações – JAX-RS

@Produces - mapeia o(s) tipo(s) MIME que será gerado como resposta de um método HTTP REST.

Quando aplicado na classe é automaticamente propagado para todos os métodos expostos como REST.

Quando aplicado no método, sobrepõe a definição da classe, especificando o MIME da resposta do método.

Anotações – JAX-RS

@Produces

1. Converte automaticamente e RETORNA os tipos: `String`, `java.io.byte[]`, `java.io.InputStream`, `Reader` e `java.io.File` na resposta HTTP.
2. Usa a API JAXB para conversão automática de objetos Java para XML e JSON.

Anotações – JAX-RS

@Consumes

1. Converte automaticamente os tipos RECEBIDOS :
String, java.io.byte[], java.io.InputStream, Reader e
java.io.File na entrada de métodos vindo na
requisição HTTP.
2. Usa a API JAXB para conversão automática de
objetos Java para XML e JSON.

Respostas com JAX-RS

1. Métodos declarados como void são mapeados automaticamente para gerar uma resposta HTTP 204 No Content (sem MIME).
2. Métodos declarados como String são mapeados automaticamente para gerar uma resposta HTTP 200 OK, convertendo o conteúdo da String no MIME declarado.

Respostas com JAX-RS

3. Métodos declarados como Objetos Java JAXB são mapeados automaticamente para gerar uma resposta HTTP 200 OK, usando MIME XML ou JSON.

4. JAX-RS fornece classes chamadas de Response e ResponseBuilder utilizadas para customizar variações de possíveis retornos, tanto relacionado com o conteúdo retornada, bem como o status HTTP.

Respostas com JAX-RS

Utilizadas para oferecer implementações de respostas dinâmicas para serviços REST. Dessa forma não precisa mais declarar `@Produces`.

Desenvolvimento de Aplicações para WEB

**Aula: Criação de APIs's
com Spring Framework.**



Prof. Anderson Augusto Bosing

Tecnologias Utilizadas

- ✓ Spring Boot
- ✓ IntelliJ
- ✓ Java 21
- ✓ Lombok
- ✓ Postgres
- ✓ JPA/Spring Data
- ✓ Maven
- ✓ Postman

Spring

Spring é um framework para desenvolvimento de aplicações em Java, criado em meados de 2002 por Rod Johnson, que se tornou bastante popular e adotado ao redor do mundo devido a sua simplicidade e facilidade de integração com outras tecnologia.

O framework foi desenvolvido de maneira modular, na qual cada recurso que ele disponibiliza é representado por um módulo, que pode ser adicionado em uma aplicação conforme as necessidades.

Spring – Conceitos Iniciais

Injeção de Dependências

É um padrão de projeto que ajuda muito a deixar o código desacoplado, melhora a legibilidade e interpretação do código, melhora a distribuição de responsabilidades entre as classes e facilita a manutenção do código.

Padrão de projetos já implementado no spring framework, este padrão possui alguns princípios e destaco dois deles:

Diminuir o acoplamento entre as classes(Trabalhar com Interfaces, se baseando em abstrações).

Alta coesão(Conceito de single responsibility).



Spring – Conceitos Iniciais

Uma dependência é simplesmente um objeto que a sua classe precisa para funcionar.

Inversão de Controle, outro padrão de projeto onde o desenvolvedor passa a responsabilidade da criação da instancia das suas classes para o próprio software ou container que implementa o padrão de injeção de dependência.

No caso do spring ele já possui a implementação do container com essa funcionalidade.



Spring – Vamos entender como utilizar esses conceitos.

Bom, agora que já conhecemos os conceitos vamos aprender como utilizar esses recursos junto ao spring framework. No spring para trabalharmos com os conceitos de DI/IOC iremos utilizar os beans.

Os beans são as classes as quais passamos a responsabilidade do gerenciamento dessas classes para o spring, e para que possamos passar a responsabilidade para ele utilizamos as anotações.

Spring – Anotações

@Component

Anotação mais básica para criar qualquer tipo de bean gerenciado pelo spring framework.

Normalmente usada quando não se define um bean como @Repository ou @Service.

```
5 @Component
6 public class ComputadorUtil {
7
8     //metodos e atributos omitidos
9
10 }
```

Spring – Anotações

@Repository

Define um bean como sendo do tipo persistente para uso de classes de acesso a banco de dados. A partir desta anotação o Spring pode usar recursos referentes a persistência, como tratar as exceções específicas para este fim.

```
5
6 @Repository
7 public class ComputadorDAO {
8
9     //metodos e atributos omitidos
10
11 }
```

Spring – Anotações

@Service

Usado para definir classes do tipo serviço(Service Layer), que possuem, por exemplo, regras de negócios.

```
0
7 @Service
8 public class ComputadorService {
9
10     //metodos e atributos omitidos
11
12 }
```

Spring – Anotações @Autowired

Anotação usada para informar ao spring que ele deve injetar a variável anotada na classe em que está declarada.

```
@Autowired  
private ComputadorService computadorService;
```

Spring – Anotações

@RestController

Usado para definir classes do tipo controller/resource.

```
@RestController  
@RequestMapping("/hello")  
public class HelloController {
```

Spring – Anotações

@PathVariable

Tem o objetivo de extrair da URL um parâmetro que foi incluído como path da URL.

```
@GetMapping(path = "/listagem/{dsMarca}")  
public String helloWorld(@PathVariable("dsMarca") String dsMarca) {  
    return "OLÁ MUNDO "+dsMarca;  
}
```

Spring – Anotações

@RequestParam

Tem o objetivo de capturar um parâmetro de consulta(Query Param) enviado por uma solicitação.

```
@GetMapping  
public String helloWorld(@RequestParam("nome") String nome) {  
    return "OLÁ MUNDO "+nome;  
}
```

Spring – Anotações

@GetMapping, @PostMapping, @PutMapping, @DeleteMapping

Usada para mapear URLs de acesso a um controller e aos métodos contidos nele definindo os verbos HTTP de acesso aos métodos.

Spring Boot

O spring boot é um projeto que chegou para facilitar o processo de configuração e publicação de nossas aplicações.

A intenção é ter o seu projeto rodando o mais rápido possível e sem complicação.


Ele consegue isso favorecendo a convenção sobre a configuração.

Com os módulos informados o spring boot vai reconhece-los e fornecer uma configuração inicial.

Basta que você diga a ele quais módulos deseja utilizar: WEB, template, persistência, segurança, etc...



Criando o projeto no Spring Initializr.



spring initializr

Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Java** ☐ Kotlin ☐ Groovy

☒ **Maven**

Language

☒ **Java** ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.1.0 (SNAPSHOT) ☐ 3.1.0 (M2) ☐ 3.0.6 (SNAPSHOT) ☐ 3.0.5

☐ 2.7.11 (SNAPSHOT) ☒ **2.7.10**

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ **Jar** ☐ War

Java ☐ 20 ☐ 17 ☒ **11** ☐ 8

Dependencies ADD DEPENDENCIES... ⌘ + B

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

PostgreSQL Driver SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.



Configurando o Banco

```
spring.datasource.url=jdbc:postgresql://localhost:5432/app  
spring.datasource.username=unipar  
spring.datasource.password=unipar123
```

