

Curso

# Aplicações JAVA com SPRING BOOT



**Prof. Msc. Antonio B. C. Sampaio Jr**  
engenheiro de software & professor

@abctreinamentos  
@amazoncodebr

[www.abctreinamentos.com.br](http://www.abctreinamentos.com.br)  
[www.amazoncode.com.br](http://www.amazoncode.com.br)



# CONTEÚDO PROGRAMÁTICO



- UNIDADE 1 – INTRODUÇÃO
- UNIDADE 2 – FUNDAMENTOS DO SPRING BOOT
- UNIDADE 3 – PERSISTÊNCIA DE DADOS NO SPRING BOOT
- UNIDADE 4 – PROJETO WEB NO SPRING BOOT
- UNIDADE 5 – PROJETO REST API NO SPRING BOOT
- UNIDADE 6 – PROJETO REST API NO SPRING BOOT COM REACTJS

## PROJETOS DO CURSO



- 1º Projeto Spring Boot – Impressão de Mensagens
- 2º Projeto Spring Boot – Impressão de Mensagens na WEB
- 3º Projeto Spring Boot – Aplicação Servidor Público
- 4º Projeto Spring Boot – Aplicação Servidor Público na WEB
- 5º Projeto Spring Boot – Aplicação Servidor Público no SGBD MYSQL
- 6º Projeto Spring Boot – Aplicação Servidor Público no MONGO DB

## PROJETOS DO CURSO



- 7º Projeto Spring Boot – Aplicação Servidor Público WEB
- 8º Projeto Spring Boot – Aplicação Servidor Público REST API e MySQL
- 9º Projeto Spring Boot – Aplicação Servidor Público REST API com REACT
- 10º Projeto Spring Boot – Aplicação Servidor Público/Curso REST API – Monolítico
- 11º Projeto Spring Boot – Aplicação Servidor Público REST API - Microserviços

## UNIDADE 5

---

### PROJETO REST API NO SPRING BOOT

API

---

WEB SERVICES

---

REST API

---

REST vs RESTFUL

---

SPRING MVC

---

PROJETO PRÁTICO

---

8º Projeto Spring Boot – Aplicação Servidor  
Público REST API e MySQL

---

API

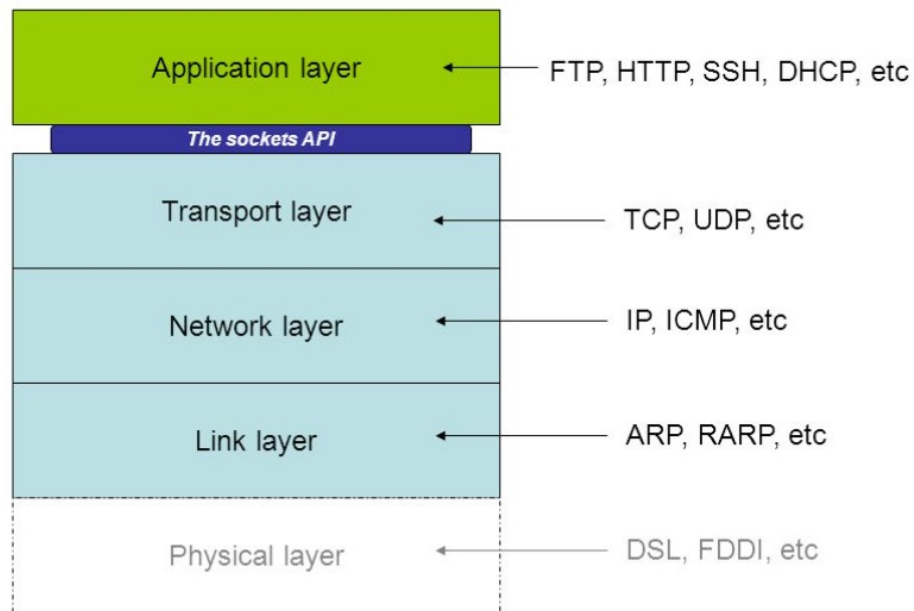
# API

---

- **O que é API?**
  - API (*Application Program Interface*) é um conjunto de definições e protocolos que permitem a comunicação entre diferentes componentes de software. Em outras palavras, uma API é uma interface que define como as diferentes partes de um sistema de software interagem umas com as outras.
  - Uma API pode ser utilizada para muitas finalidades diferentes, como permitir que diferentes sistemas ou aplicativos se comuniquem, fornecer acesso programático a recursos e serviços em uma plataforma, definir um conjunto de operações padronizadas que podem ser usadas por desenvolvedores para interagir com um sistema, entre outras.

# API

- Arquitetura TCP/IP

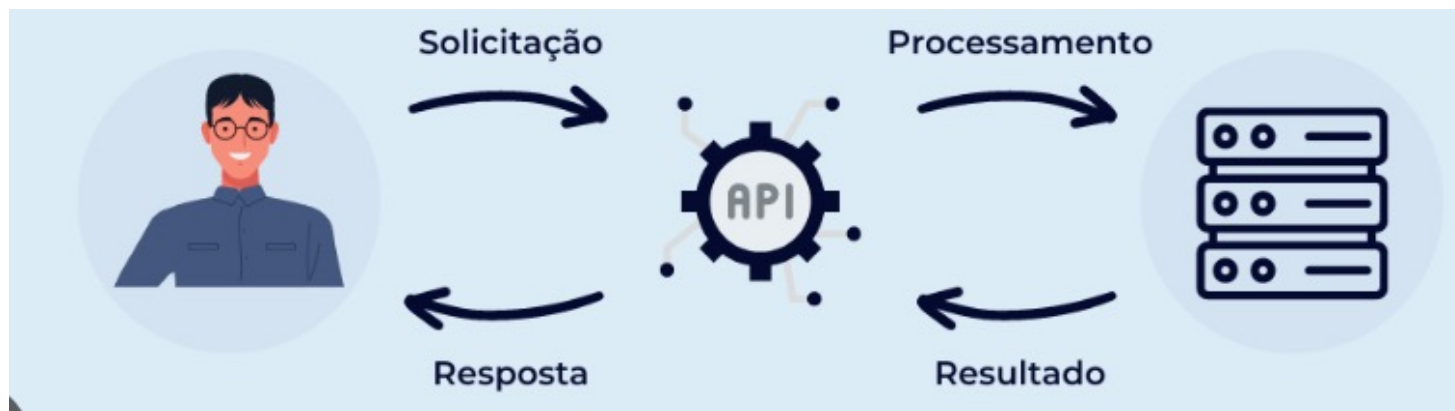


- A **interface sockets API** pode ser pensada como um contrato de serviço entre as duas camadas: **Aplicação** e **Transporte**. Esse contrato define como essas duas camadas se comunicam trocando solicitações e respostas.



# TIPOS DE APIs

- Existem quatro maneiras diferentes pelas quais as APIs podem funcionar:
  - (1) **APIs baseadas em solicitação e resposta:** Neste tipo de API, o cliente faz uma solicitação para o servidor através de uma chamada de API e o servidor retorna uma resposta. As APIs baseadas em solicitação e resposta podem usar vários protocolos, como **HTTP/REST**, SOAP/XML, gRPC/protobuf, entre outros.



<https://mambowifi.com/api-x-webhook-qual-a-diferenca-e-em-quais-casos-usar/>

# TIPOS DE APIs

- Existem quatro maneiras diferentes pelas quais as APIs podem funcionar:
  - (1) APIs baseadas em solicitação e resposta:  
HTTP/REST



## REST REQUEST

GET https://sample.com/person/1



## REST JSON

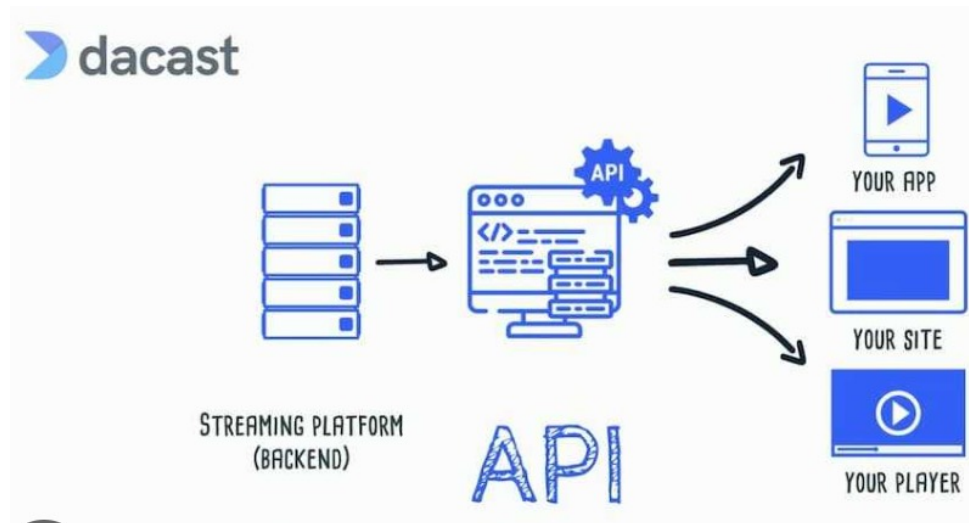
```
{
  "firstName": "John",
  "middleName": "Andrew",
  "lastName": "Smith",
  "email": "jas1992@gmail.com",
  "relationshipStatus": "single"
}
```

<https://research.aimultiple.com/graphql-vs-rest/>

# TIPOS DE APIs

---

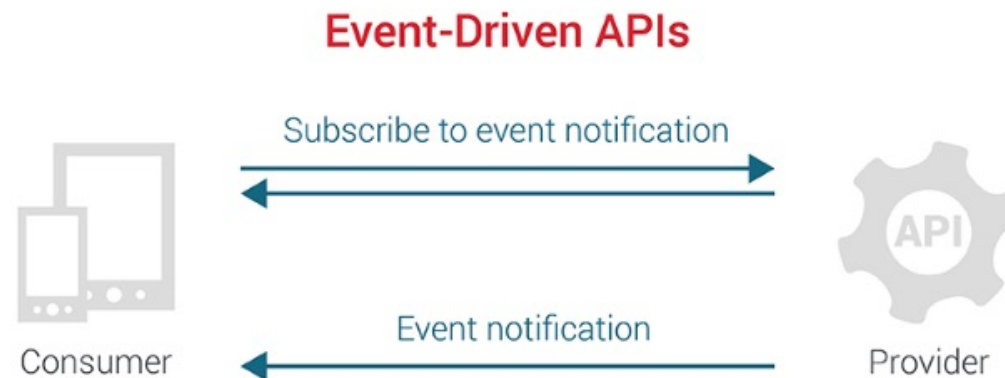
- Existem quatro maneiras diferentes pelas quais as APIs podem funcionar:
  - **(2) APIs baseadas em streaming:** Neste tipo de API, o cliente pode enviar ou receber dados em tempo real por meio de um fluxo contínuo de dados. As APIs baseadas em streaming são usadas em aplicativos que exigem comunicação em tempo real, como bate-papo em tempo real, transmissão de vídeo, jogos on-line, entre outros.



# TIPOS DE APIs

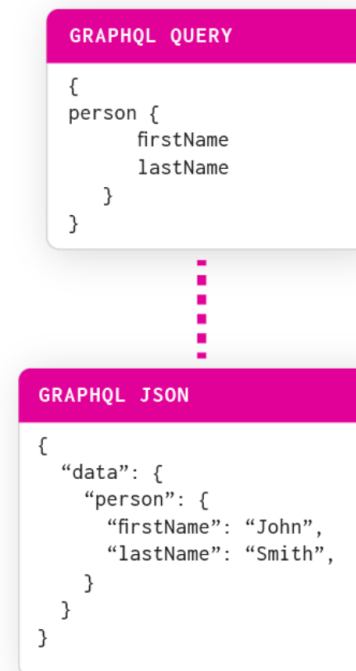
---

- Existem quatro maneiras diferentes pelas quais as APIs podem funcionar:
  - **(3) APIs baseadas em eventos:** Neste tipo de API, o servidor envia uma mensagem ao cliente sempre que um evento ocorre. As APIs baseadas em eventos são usadas para notificar os clientes de alterações em dados, como novas mensagens em um fórum de discussão ou um novo pedido em um sistema de comércio eletrônico.



# TIPOS DE APIs

- Existem quatro maneiras diferentes pelas quais as APIs podem funcionar:
  - **(4) APIs baseadas em consulta:** Neste tipo de API, o cliente pode enviar consultas complexas para o servidor, que retorna os resultados da consulta. As APIs baseadas em consulta são usadas para buscar informações de um banco de dados ou para realizar operações em dados existentes, como atualizar ou excluir registros.



<https://research.aimultiple.com/graphql-vs-rest/>

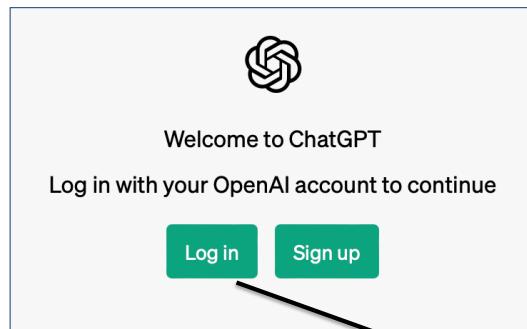
# ESCOPO DAS APIs

---

- **São 04 os Escopos de Uso**
  - **APIs Privadas**
    - Elas são internas a uma empresa e são usadas apenas para conectar sistemas e dados dentro da empresa.
  - **APIs Públicas**
    - Estas são abertas ao público e podem ser usadas por qualquer pessoa. Pode ou não haver alguma autorização e custo associado a esses tipos de APIs.
  - **APIs de Parceiros**
    - Estas são acessíveis apenas por desenvolvedores externos autorizados para auxiliar as parcerias entre empresas.
  - **APIs Compostas**
    - Estas combinam duas ou mais APIs distintas para atender a requisitos ou comportamentos complexos do sistema.

# ESCOPO DAS APIs

- APIs de Parceiros



## Welcome back

Email address



Continue

Don't have an account? [Sign up](#)

OR



Continue with Google



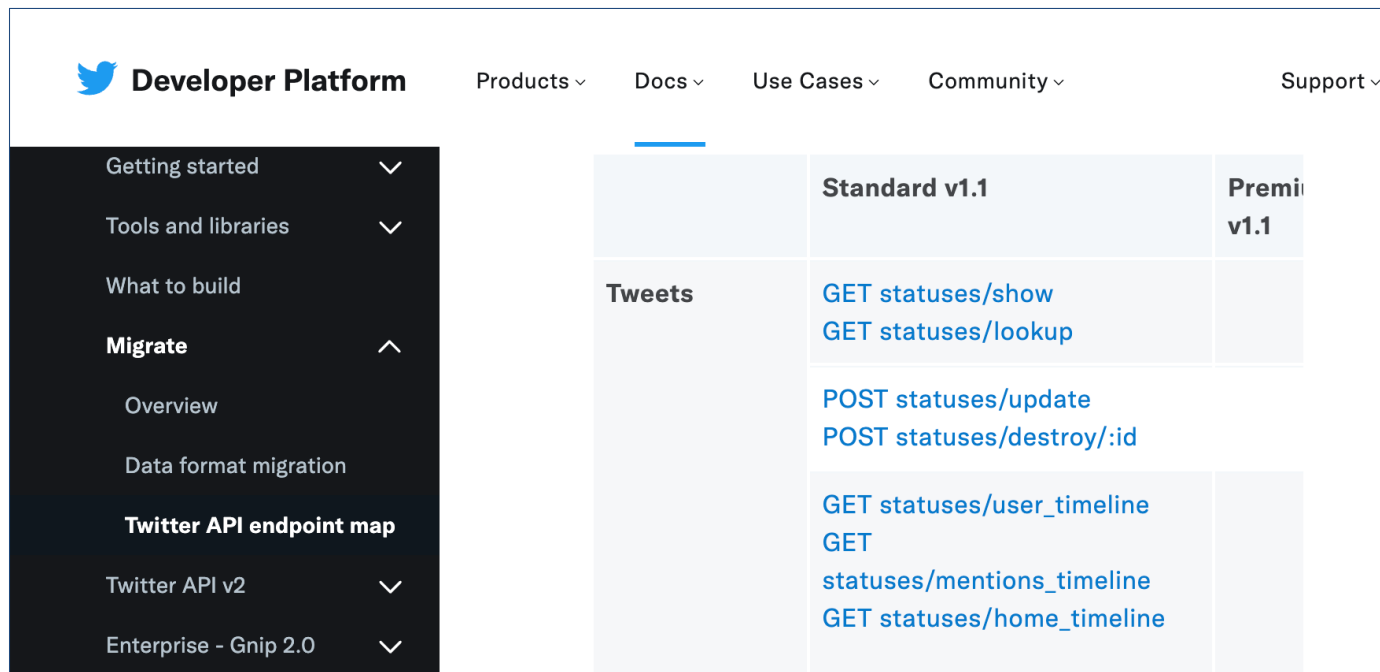
Continue with Microsoft Account

API GOOGLE

API MICROSOFT

# ESCOPO DAS APIs

- APIs Públicas



The screenshot shows the Twitter Developer Platform documentation page. The left sidebar contains a navigation menu with the following items: 'Getting started', 'Tools and libraries', 'What to build', 'Migrate' (highlighted), 'Overview', 'Data format migration', 'Twitter API endpoint map' (highlighted), 'Twitter API v2', and 'Enterprise - Gnip 2.0'. The main content area displays a table titled 'Twitter API endpoint map' with columns for 'Products', 'Docs', 'Use Cases', 'Community', and 'Support'. The table lists various API endpoints and their corresponding products.

	Products	Docs	Use Cases	Community	Support
		<b>Standard v1.1</b>			<b>Premium v1.1</b>
<b>Tweets</b>		<a href="#">GET statuses/show</a> <a href="#">GET statuses/lookup</a>			
		<a href="#">POST statuses/update</a> <a href="#">POST statuses/destroy/:id</a>			
		<a href="#">GET statuses/user_timeline</a> <a href="#">GET statuses/mentions_timeline</a> <a href="#">GET statuses/home_timeline</a>			

<https://developer.twitter.com/en/docs/twitter-api/migrate/twitter-api-endpoint-map>



# ESCOPO DAS APIs

- APIs Públicas

Meta for Developers

DocumentosFerramentasSuporte

Pesquisar documentação do desenvolvedor

API Endpoints

Nesta Página

All endpoints start with <https://graph.facebook.com/v3.3>

Endpoint	Description
<a href="#">GET /&lt;page-id&gt;/commerce_orders</a>	List all commerce orders associated with a page
<a href="#">GET /&lt;order-id&gt;</a>	Fetch order details for a given order ID
<a href="#">GET /&lt;order-id&gt;/items</a>	Fetch line items for a given order ID
<a href="#">GET /&lt;order-id&gt;/shipments</a>	Fetch all shipments for a given order ID

<https://developers.facebook.com/docs/commerce-platform/order-management/api-endpoints/>

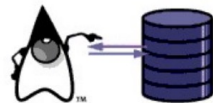
# APIs e Web Services

---

- **Diferenças entre APIs e Web Services**

- **APIs** e **Web Services** são dois conceitos relacionados, mas diferentes. As APIs definem um conjunto de operações e funcionalidades que podem ser acessadas e utilizadas por outros programas. Não necessitam de comunicação via rede para funcionarem.

- **Exemplo:** API JDBC



- Os Web Services são APIs que precisam enviar e receber dados pela rede.

# APIs e Web Services

---

- **Diferenças entre APIs e Web Services**
  - Webservices são um tipo de API que utilizam tecnologias da web para permitir a comunicação entre diferentes sistemas, plataformas e linguagens de programação. Existem diferentes tipos de Web Services, como SOAP/XML e REST/JSON, que utilizam diferentes protocolos e formatos de dados para a comunicação.
  - Resumindo, enquanto APIs são interfaces de programação que podem ser utilizadas em diferentes tipos de aplicações, Web Services são um tipo de API que utiliza tecnologias da web para permitir a comunicação entre diferentes sistemas e plataformas.

WEB SERVICES

# WEB SERVICES

- **Definição**
  - Os **Web Services** são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML.
  - Principais padrões: **SOAP/XML** e **HTTP/JSON (REST)**.



# WEB SERVICES

---

- Padrão SOAP/XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org
/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap
/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi
="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http
://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:MinhaUFLA.authResponse xmlns:ns1="urn:minhaufla">
      <return xsi:type="xsd:string">{"&quot;id&quot;:&quot;800_1
      .4&quot;,&quot;message&quot;:&quot;ST-119242
      -8CvOxadIu9HBEbNZD6sk-casdgti&quot;,&quot;type&quot;
      ;:&quot;SUCESSO&quot;,&quot;system&quot;:&quot;
      ;MINHAUFLA&quot;}</return>
    </ns1:MinhaUFLA.authResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo de mensagem SOAP-XML

# WEB SERVICES

---

- Padrão HTTP/JSON (Padrão REST)

```
{  
  "id": "800_1.4",  
  "message": "131vnt2td8h1hhnaens1d7jla1",  
  "type": "SUCESSO",  
  "system": "MINHAUFLA"  
}
```

Exemplo de mensagem REST-JSON

# WEB SERVICES

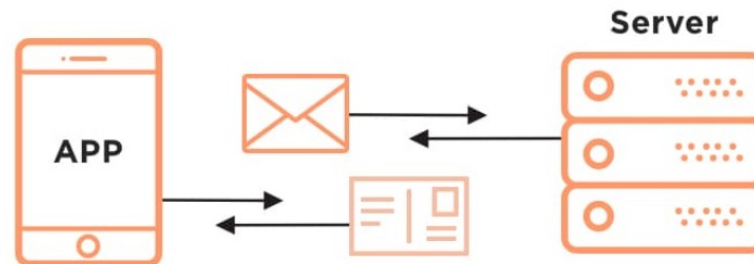
---

- Comparação SOAP/XML & HTTP/JSON (REST)

## SOAP vs. REST APIs

### SOAP IS LIKE USING AN ENVELOPE

Extra overhead, more bandwidth required, more work on both ends(sealing and opening).



### REST IS LIKE A POSTCARD

Lighterweight, can be cached, easier to update

Source: <https://assets-global.website-files.com>



REST API

# REST API

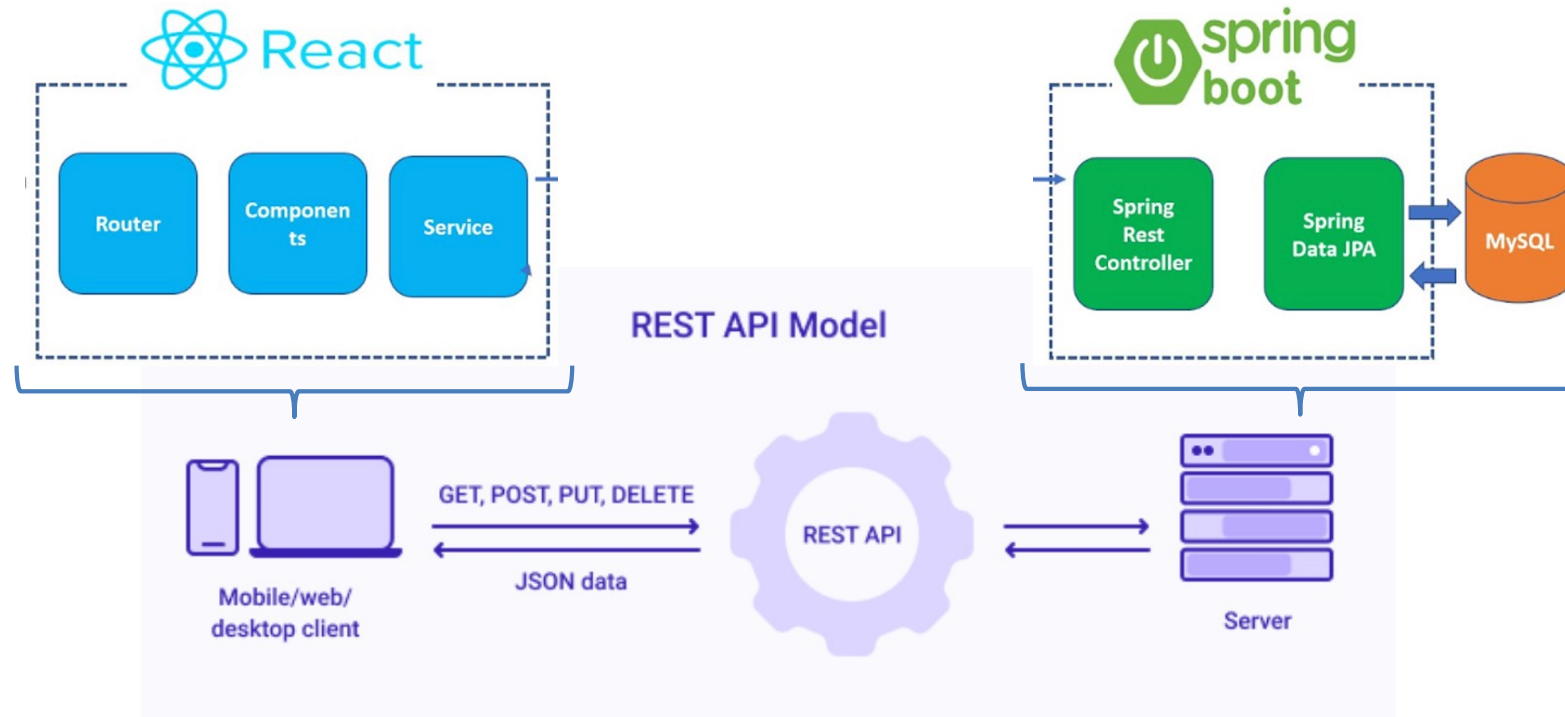
---

- **Definição**

- REST é um formato de WebServices que simplifica o uso de recursos computacionais, tais como capacidade de processamento dos dispositivos, bem como o consumo de banda de dados nas redes de comunicações, algo imperativo com a “explosão” de uso dos dispositivos móveis.
- Todo REST é um Web Service, mas nem todo Web Service é um REST.
- Com o uso da Arquitetura REST, há uma total desvinculação do back-end com o front-end.

# REST API

- Front-End & Back-End



# REST API

---

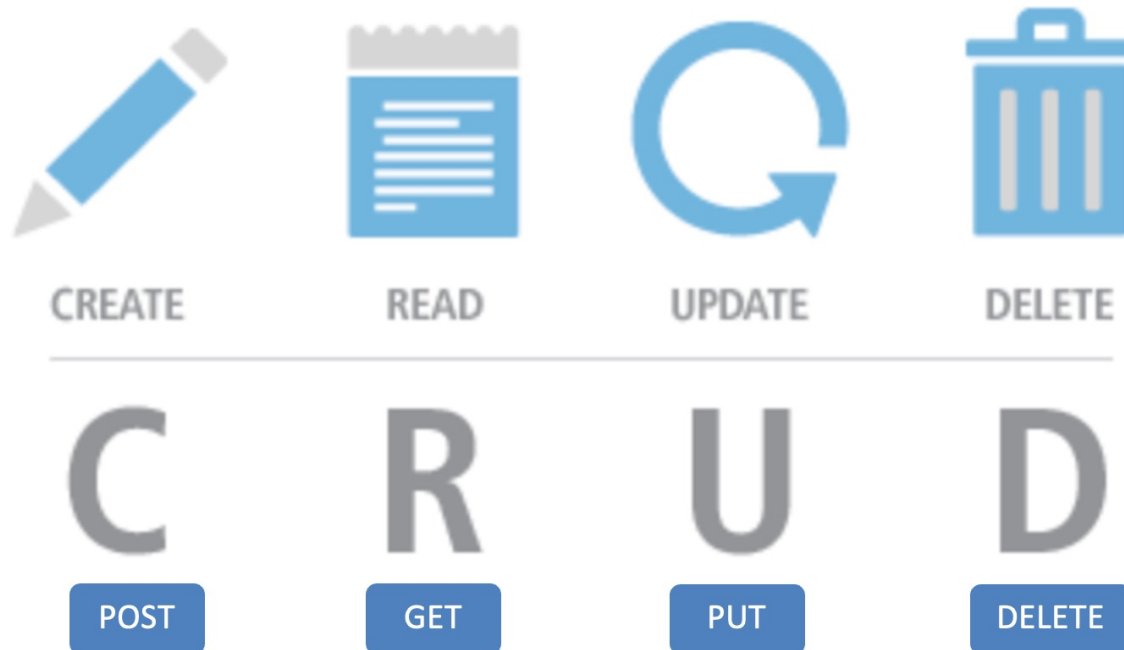
- **Arquitetura**

- **REST** é uma Arquitetura que define um conjunto de boas práticas para a especificação e construção de APIs. Por isso, essas APIs devem ser desenvolvidas dentro das melhores práticas.
- Utiliza o protocolo **HTTP** para a troca de mensagens (requisição/resposta) e o **JSON** para o envio de dados.
- Pode ser desenvolvida em qualquer tecnologia (Java, JS, Python, PHP, C#, Ruby, etc.)
- Utiliza verbos HTTP para a realização de operações CRUD.

# REST API

---

- Operações CRUD



# REST API

---

- **Principais Princípios**
  - **(1) Cliente-Servidor**
    - A separação entre cliente e servidor permite que cada um evolua independentemente, com baixo acoplamento entre eles. O cliente não precisa saber como as informações são processadas no servidor e o servidor não precisa saber como o cliente apresenta os dados.
  - **(2) Stateless**
    - Cada solicitação enviada pelo cliente contém todas as informações necessárias para executar a operação solicitada, sem depender de solicitações anteriores. Isso permite que o servidor seja escalável, já que não precisa manter o estado da sessão do cliente.

# REST API

---

- **Principais Princípios**

- **(3) Cacheable**

- As respostas das solicitações podem ser armazenadas em cache para melhorar o desempenho e a escalabilidade. Isso permite que os clientes reutilizem as respostas armazenadas em cache, evitando solicitações desnecessárias ao servidor.

- **(4) Uniform Interface**

- O cliente e o servidor usam uma interface uniforme para se comunicar, o que simplifica a integração entre diferentes sistemas e aplicativos.

# REST API

---

- **(4) Uniform Interface**

- A interface uniforme é composta pelos seguintes elementos:
  - **Identificação de recursos:** Cada recurso é identificado por um URI exclusivo.
  - **Manipulação de recursos através de representações:** As solicitações e respostas são feitas através de representações de recursos, como JSON.
  - **Mensagens auto descritivas:** As mensagens de solicitação e resposta contêm informações suficientes para que o receptor entenda o significado da mensagem.
  - **HATEOAS (*Hypermedia as the Engine of Application State*):** As respostas devem conter links que permitam a navegação pelo serviço. Isso permite que o cliente descubra e acesse recursos relacionados sem conhecimento prévio do serviço.



# REST API

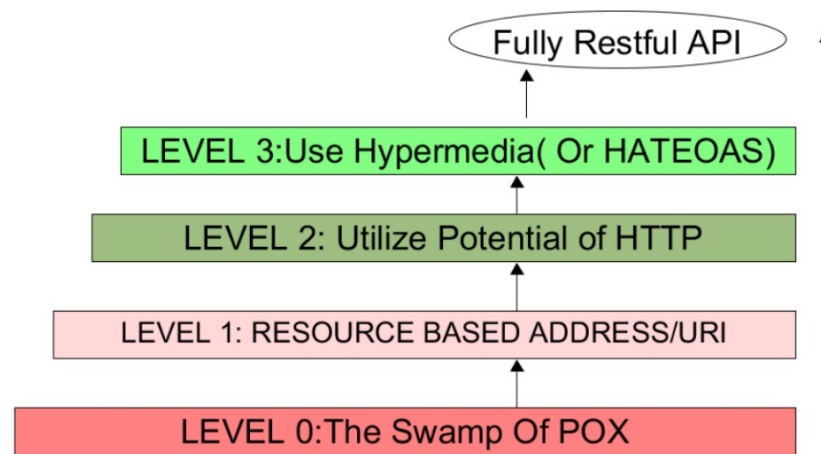
---

- **Principais Princípios**
  - (5) **Sistema em camadas:** O sistema pode ser composto de várias camadas, onde cada camada só precisa se comunicar com as camadas adjacentes. Isso permite que o sistema seja escalável e flexível.

REST vs  
RESTFUL

# REST vs RESTFUL

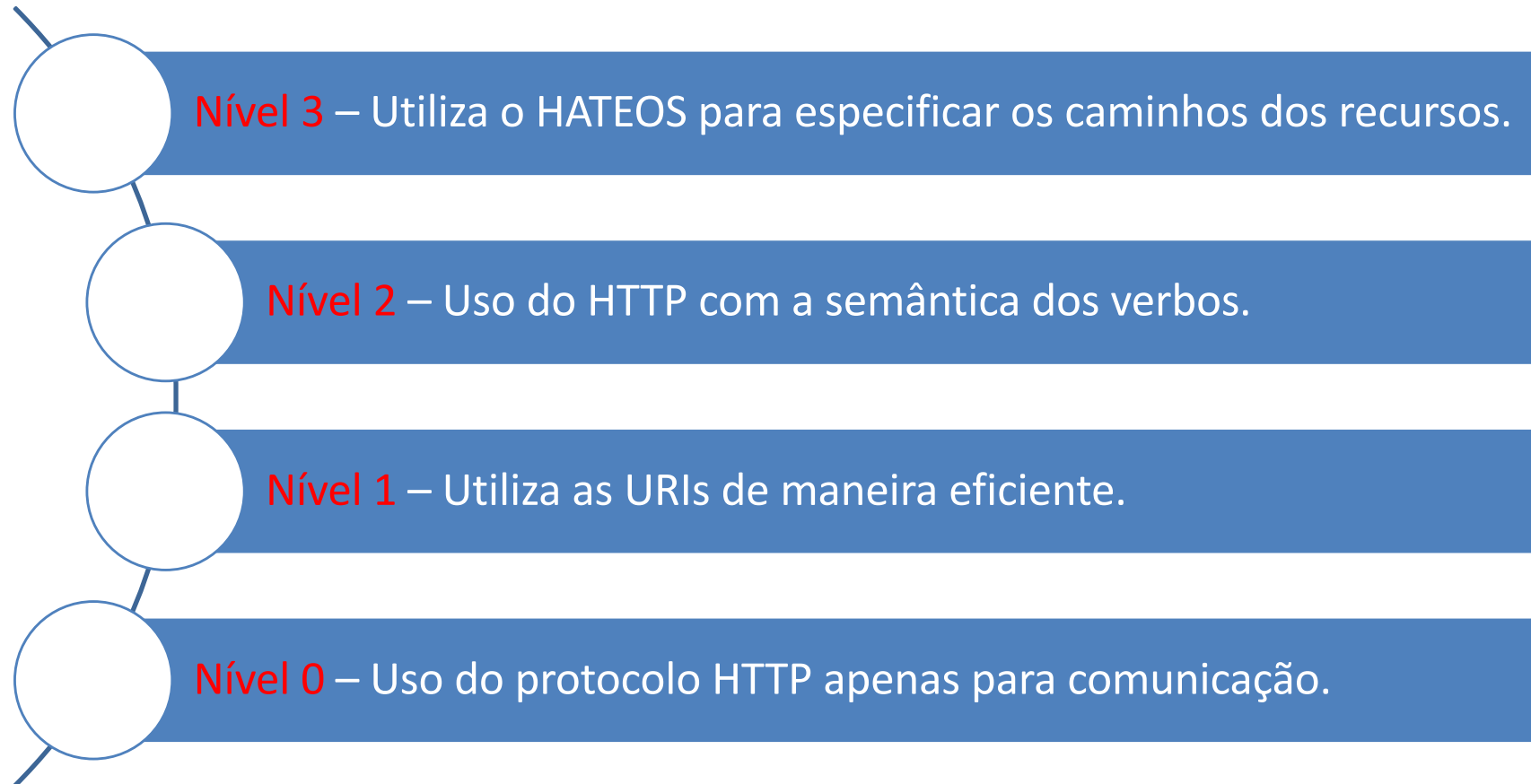
- **REST** é um conceito arquitetural para implementar APIs seguindo as melhores práticas (Conceito Abstrato).
- **RESTFUL** é a implementação do conceito REST.
- E como saber se o Web Service dito **RESTFUL** realmente implementa o conceito REST?



Modelo de Maturidade de Leonard Richardson

# MODELO DE MATURIDADE

---



SPRING MVC

# SPRING MVC

---

- Definição

- O **Spring MVC** é um módulo do Spring Framework que é adicionado pelo starter **spring-boot-starter-web**.

spring-boot-starter-web

```
<artifactId>  
    spring-boot-starter-web  
</artifactId>
```

◀ Spring MVC

◀ REST 

◀ Tomcat

◀ Jackson

# SPRING MVC

- O **spring-boot-starter-web** é uma das dependências principais do Spring Boot que oferece suporte para a criação de aplicativos web. Ele fornece recursos para desenvolver aplicativos web, incluindo o suporte para a criação de endpoints RESTful.

Meta for Developers

DocumentosFerramentasSuporte

Pesquisar documentação do desenvolvedor

API Endpoints

Nesta Página

All endpoints start with <https://graph.facebook.com/v3.3>

Endpoint	Description
<a href="#">GET /&lt;page-id&gt;/commerce_orders</a>	List all commerce orders associated with a page
<a href="#">GET /&lt;order-id&gt;</a>	Fetch order details for a given order ID
<a href="#">GET /&lt;order-id&gt;/items</a>	Fetch line items for a given order ID
<a href="#">GET /&lt;order-id&gt;/shipments</a>	Fetch all shipments for a given order ID



# ANOTAÇÕES SPRING MVC

---

- Principais Anotações

- **@RestController**: é uma anotação utilizada na classe para indicar que ela é um controlador REST, que lida com as solicitações HTTP enviadas para a API. Ela combina as anotações @Controller e @ResponseBody, permitindo o retorno de objetos Java diretamente como resposta, ao invés de modelos.

```
@RestController  
public class ServidorPublicoController {  
    //métodos para receber as requisições  
}
```

- Uma aplicação que usa a anotação **@RestController** para criar endpoints RESTful **não é compatível com o Thymeleaf**. Isso ocorre porque, com a anotação @RestController, as respostas são geradas como dados serializados em vez de HTML dinâmico.





# ANOTAÇÕES SPRING MVC

---

- Principais Anotações

- **@RequestMapping**: é uma anotação utilizada para mapear uma solicitação HTTP para um método específico em um controlador. É possível definir o caminho da solicitação e o método HTTP que o método do controlador irá lidar.
- **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping**: essas anotações são usadas para mapear as solicitações HTTP GET, POST, PUT e DELETE, respectivamente. Elas são uma maneira mais conveniente de usar a anotação **@RequestMapping**.

```
@RestController
public class ServidorPublicoController {
    @GetMapping("/")
    public String alo(){
        return "Alo 😊";
    }
}
```



# ANOTAÇÕES SPRING MVC

---

- **Principais Anotações**

- **@RequestBody**: essa anotação é usada para mapear o corpo da solicitação para um objeto Java em um método de controlador. Ela é usada quando o cliente envia dados para a API em um formato como JSON ou XML.

```
@RequestMapping(method=RequestMethod.POST, consumers= "application/json")  
public @ResponseBody Course saveCourse(@RequestBody Course aCourse)  
{  
    return courseRepository.save(aCourse);  
}
```



# ANOTAÇÕES SPRING MVC

---

- Principais Anotações

- **@ResponseStatus**: essa anotação é usada para definir o código de status da resposta HTTP. Ela é útil para personalizar os códigos de status de acordo com o resultado da operação.

```
@RequestMapping(method=RequestMethod.POST, consumers= "application/json")  
public @ResponseBody Course saveCourse(@RequestBody Course aCourse)  
{  
    return courseRepository.save(aCourse);  
}
```



# ANOTAÇÕES SPRING MVC

---

- Principais Anotações

- **@ExceptionHandler**: essa anotação é usada para capturar exceções lançadas pelo controlador que são geradas por métodos anotados com @RequestMapping, @GetMapping, @PostMapping, etc.

```
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<Object> handleNotFoundException(ResourceNotFoundException ex) {
    Map<String, Object> body = new LinkedHashMap<>();
    body.put("message", ex.getMessage());
    body.put("status", HttpStatus.NOT_FOUND);
    return new ResponseEntity<>(body, HttpStatus.NOT_FOUND);
}
```

# RESPONSEENTITY

---

- **Definição**

- **ResponseEntity** é uma classe da biblioteca Spring Framework em Java, que representa uma resposta HTTP de um controlador de uma API REST. Ele encapsula a resposta HTTP completa, incluindo o código de status, cabeçalhos e corpo da resposta.
- Usar o **ResponseEntity** permite ter mais controle sobre a resposta HTTP que está sendo enviada de volta ao cliente.

```
@GetMapping("/exemplo")
public ResponseEntity<String> exemplo() {
    String resposta = "Exemplo de resposta personalizada";
    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "text/plain");
    return new ResponseEntity<>(resposta, headers, HttpStatus.OK);
}
```

# CÓDIGOS HTTP

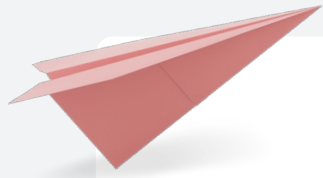
---

- **Definição**

- São códigos numéricos que indicam o status da resposta de uma solicitação HTTP.

<b>Information</b> [ 100 - 199 ] <b>100</b> - Continue <b>101</b> - Switching Protocols <b>102</b> - Processing <b>103</b> - Early hints	<b>Success</b> [ 200 - 299 ] <b>200</b> - Ok <b>201</b> - Created <b>202</b> - Accepted <b>204</b> - No Content <b>206</b> - Partial Content	<b>Redirect</b> [ 300 - 399 ] <b>300</b> - Multiple choices <b>301</b> - Moved Permanently <b>304</b> - Not Modified <b>307</b> - Temporary redirect <b>308</b> - Permanent redirect	<b>Client Error</b> [ 400 - 499 ] <b>400</b> - Bad request <b>401</b> - Unauthorized <b>403</b> - Forbidden <b>404</b> - Not found <b>409</b> - Conflict
<b>Server Error</b> [ 500 - 599 ] <b>500</b> - Internal server error <b>501</b> - Not implemented <b>502</b> - Bad gateway <b>503</b> - Service unavailable <b>504</b> - Gateway timeout			

PROJETO PRÁTICO



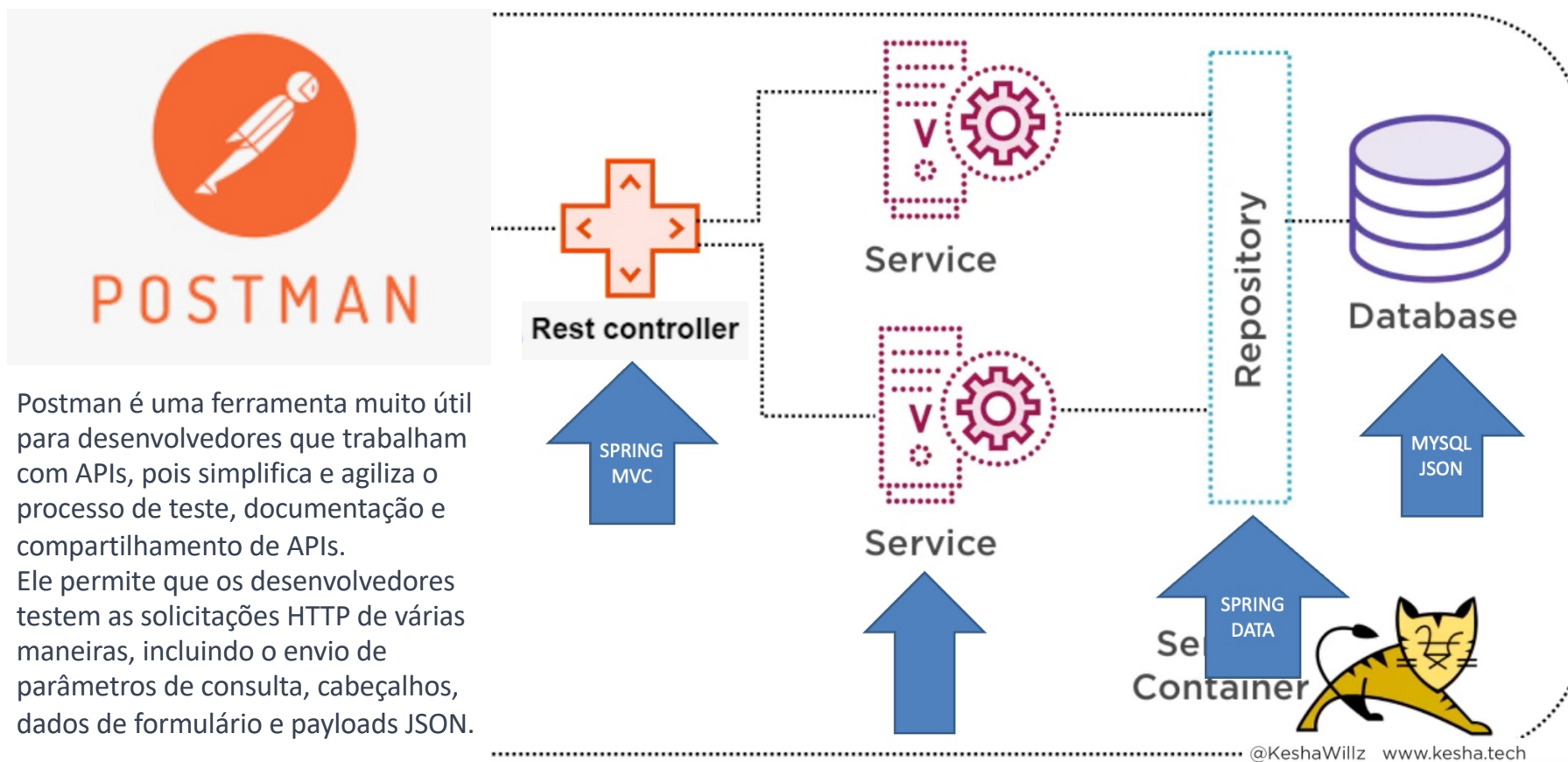
# 8º Projeto Spring Boot – Aplicação Servidor Público REST API e MySQL

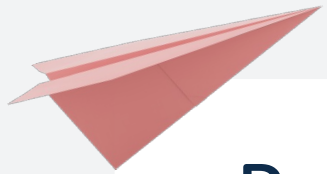






# ARQUITETURA SPRING BOOT

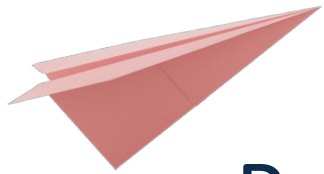




## Passos:

- Copiar o projeto anterior (Web) e fazer os ajustes necessários
- Criar a API ServidorpublicoAPIRest
- Implementar esta API
- Instalar o Postman (Cliente)





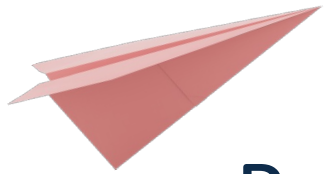
# Passos:

- Instalar o Postman (Cliente)

The screenshot shows the Postman website's download page. At the top, there's a navigation bar with links for Product, Pricing, Enterprise, Resources and Support, and Explore, along with a 'Launch Postman' button. The main heading is 'Download Postman', followed by a paragraph: 'Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.' Below this, the section 'The Postman app' is shown, with a description: 'Download the app to get started with the Postman API Platform.' Two buttons are provided: 'Mac Intel Chip' and 'Mac Apple Chip'. To the right, a snippet of the Postman application interface is visible, showing a workspace named 'Notion's Public Workspace' with a collection 'Notion API' containing 'Users' and 'Databases'. A request is being made to 'Retrieve a database' using a GET method to the URL 'https://api.notion.com/v1/databases/id'.

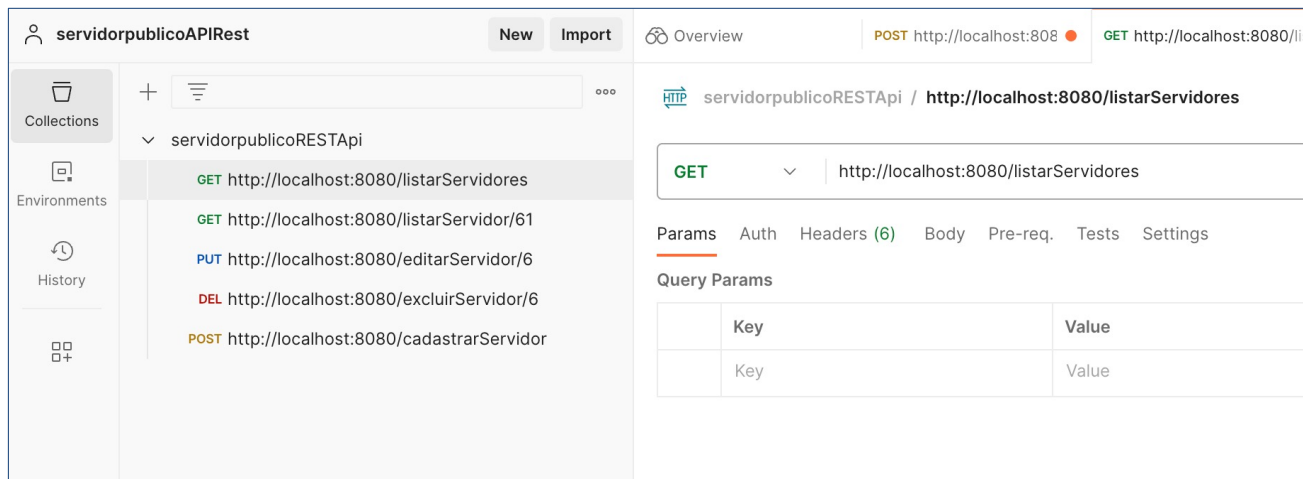
<https://www.postman.com/downloads/>





## Passos:

- Criar as chamadas à APIRest no Postman (Cliente)





# ARQUITETURA DO PROJETO

---

