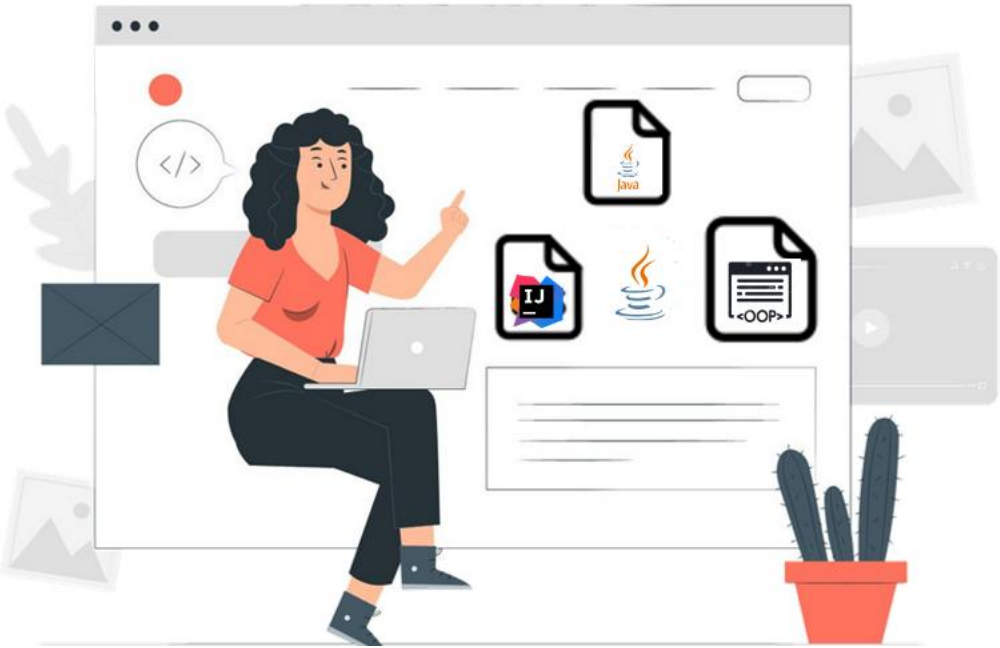


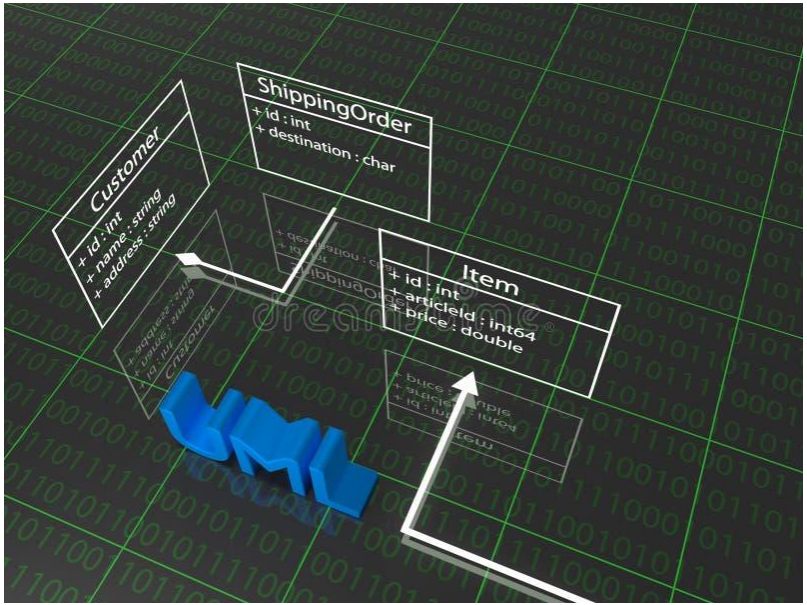
Domain Driven Design

Eliane Marion

FIAP

AGENDA DE TRABALHO

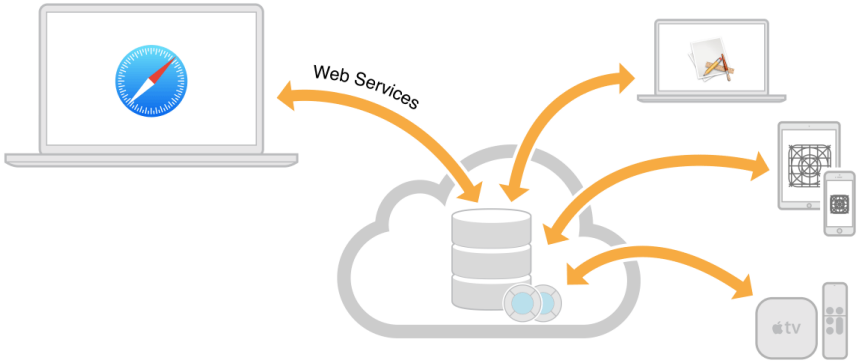
INTEGRAÇÃO
WEB



01

WEB SERVICES

WEB SERVICES

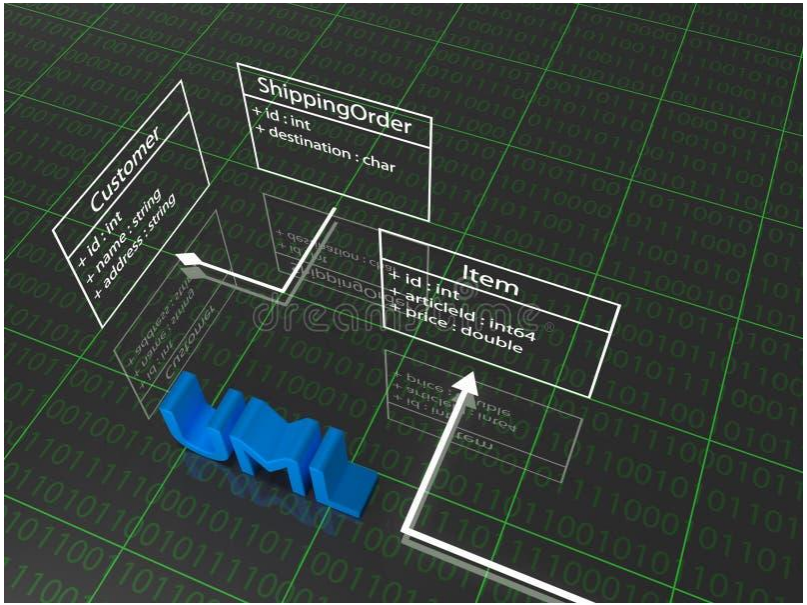


Integração e comunicação entre sistemas diferentes;

Independente de plataforma (Java, .NET, PHP, Ruby, etc..)

Permite o envio e recebimento de dados em formatos XML, Json, CSV, etc..

Os dois tipos mais comuns de web services: SOAP e REST;


ShippingOrder
+ id : int
+ destination : char
Customer
+ id : int
+ name : string
+ address : string
Item
+ id : int
+ articleId : int64
+ price : double
UML

02

REST API

REST

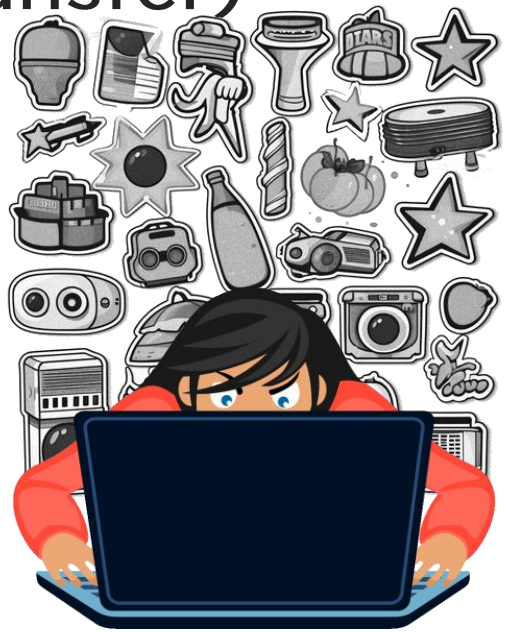
X

RESTful

REST: define um conjunto de regras e boas práticas para o desenvolvimento de APIs que possibilitam a execução de solicitações e o recebimento de respostas via protocolo HTTP, como GET e POST, permitindo a comunicação entre aplicações.

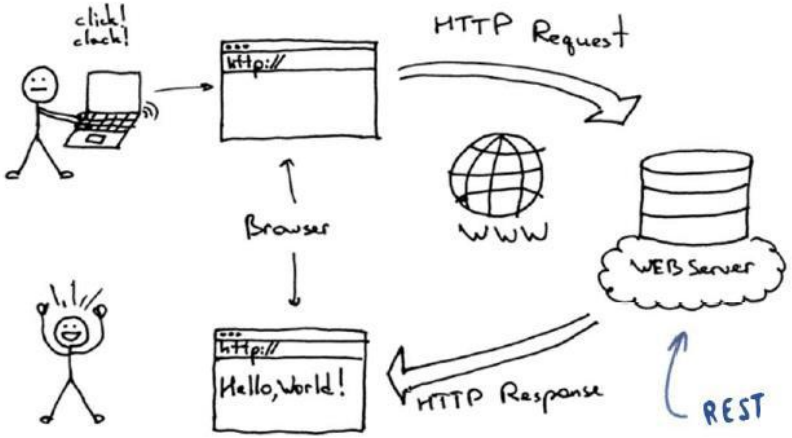
RESTFul: Existe uma confusão entre os termos, entretanto a diferença é apenas gramatical. Sistemas que utilizam os princípios Rest são chamado **RESTFul**.

RESTFul (REpresentational State Transfer)



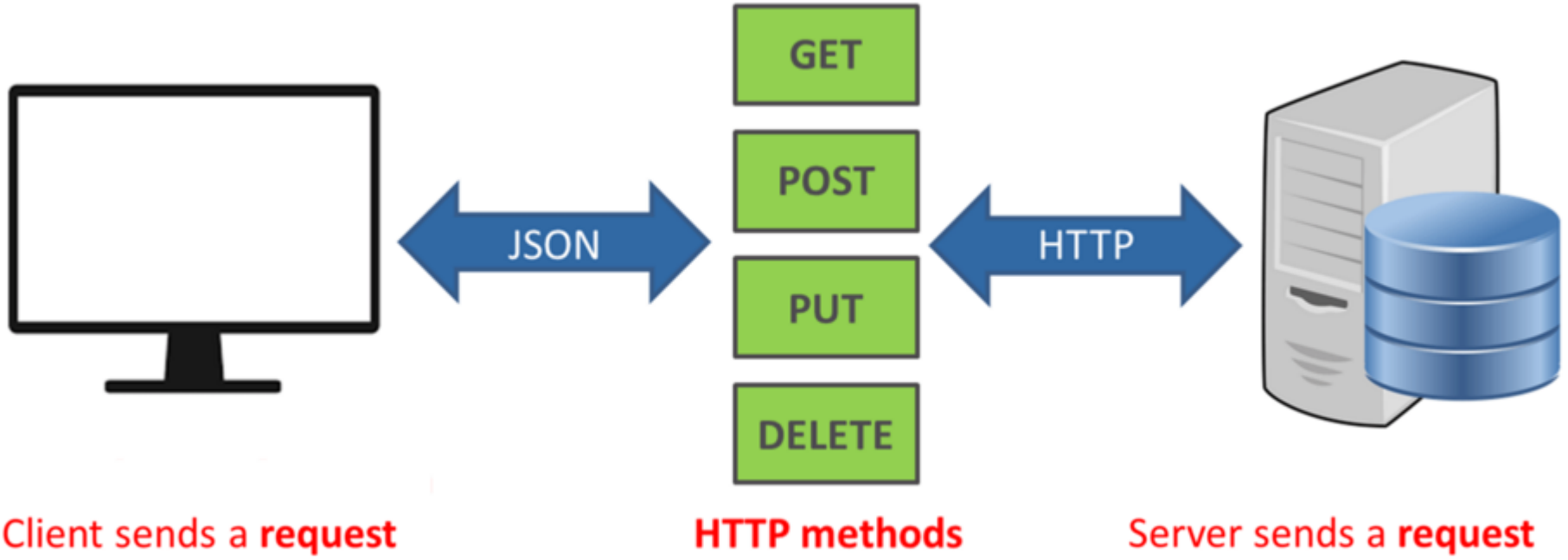
- Simples, leve, fácil de desenvolver e evoluir;
- Tudo é um recurso (Resource);
- Cada recurso possui um identificador (URI);
- Recursos podem utilizar vários formatos: html, xml, Json;
- Utiliza o Protocolo HTTP;
- Os métodos HTTP: GET, POST, PUT, DELETE

REST - Funcionamento



Toda a comunicação da interface REST é feita via web, ou seja, através de uma requisição (pedido feito pelo cliente) a uma URI (Uniform Resource Identifier), que referência um recurso, utilizando um dos quatro métodos HTTP (GET, PUT, POST ou DELETE) que, por sua vez, traz uma resposta.

REST


GET
POST
PUT
DELETE
JSON
HTTP
Client sends a **request**
HTTP methods
Server sends a **request**

URI – Unified Resource Identifier

Quando realizamos uma requisição, é preciso determinar o **endereço** do **recurso** que vamos

VERBO	URI	AÇÃO
POST	/contato/	Cria um novo recurso
GET	/contato/1	Visualizar / Recupera informações de um recurso
PUT	/contato/1	Alterar / Atualiza um recurso
DELETE	/contato/1	Apagar / Remove um recurso

HTTP STATUS CODE

- 1xx – Informativa;
- 2xx – Sucesso;
- 3xx – Redirecionamentos;
- 4xx – Erros do cliente;
- 5xx – Erros do servidor;

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi corretamente concluída. As respostas são agrupadas em cinco classes: respostas de informação, respostas de sucesso, redirecionamentos, erros do cliente e erros do servidor.

HTTP STATUS CODE

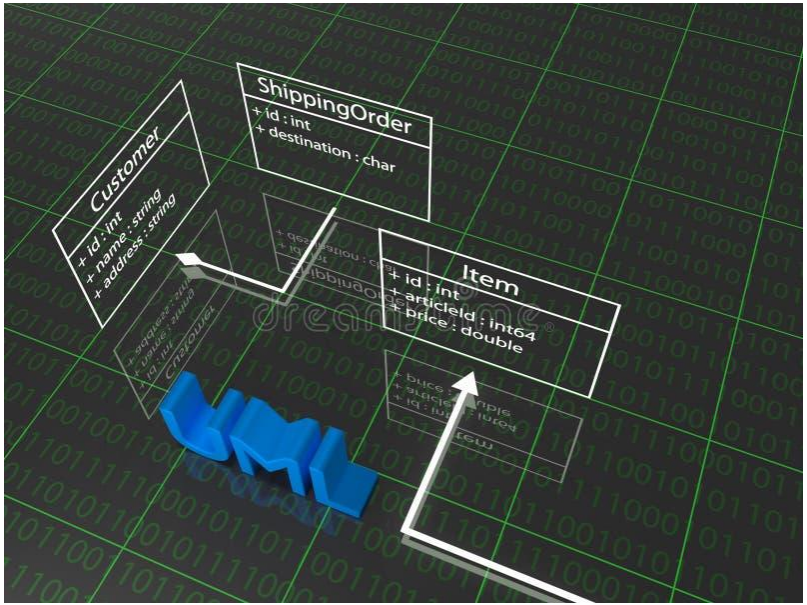
- **1xx** – Informativa;
- **2xx** – Sucesso;
- **3xx** – Redirecionamentos;
- **4xx** – Erros do cliente;
- **5xx** – Erros do servidor;

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi corretamente concluída. As respostas são agrupadas em cinco classes: respostas de informação, respostas de sucesso, redirecionamentos, erros do cliente e erros do servidor.

HTTP STATUS CODE

- Vamos trabalhar com alguns códigos na implementação do **Web**

CODE	DESCRIÇÃO
200	Ok
201	Created (Criado)
204	No Content (Sem conteúdo)
500	Internal Server Error
404	Not Found
405	Method not Allowed


ShippingOrder
+ id : int
+ destination : char
Customer
+ id : int
+ name : string
+ address : string
Item
+ id : int
+ articleId : int64
+ price : double
UML

03

JAX-RS

Introdução ao **JAX-RS**

No JAX-RS (Java API for RESTful Web Services), um "resource" é um componente chave que é modelado para corresponder a um recurso no mundo real que pode ser manipulado via HTTP. Aqui está o passo a passo para entender como um recurso funciona no JAX-RS:

ANOTAÇÕES DE RECURSO

O JAX-RS utiliza anotações para definir recursos e mapear métodos Java para operações HTTP.

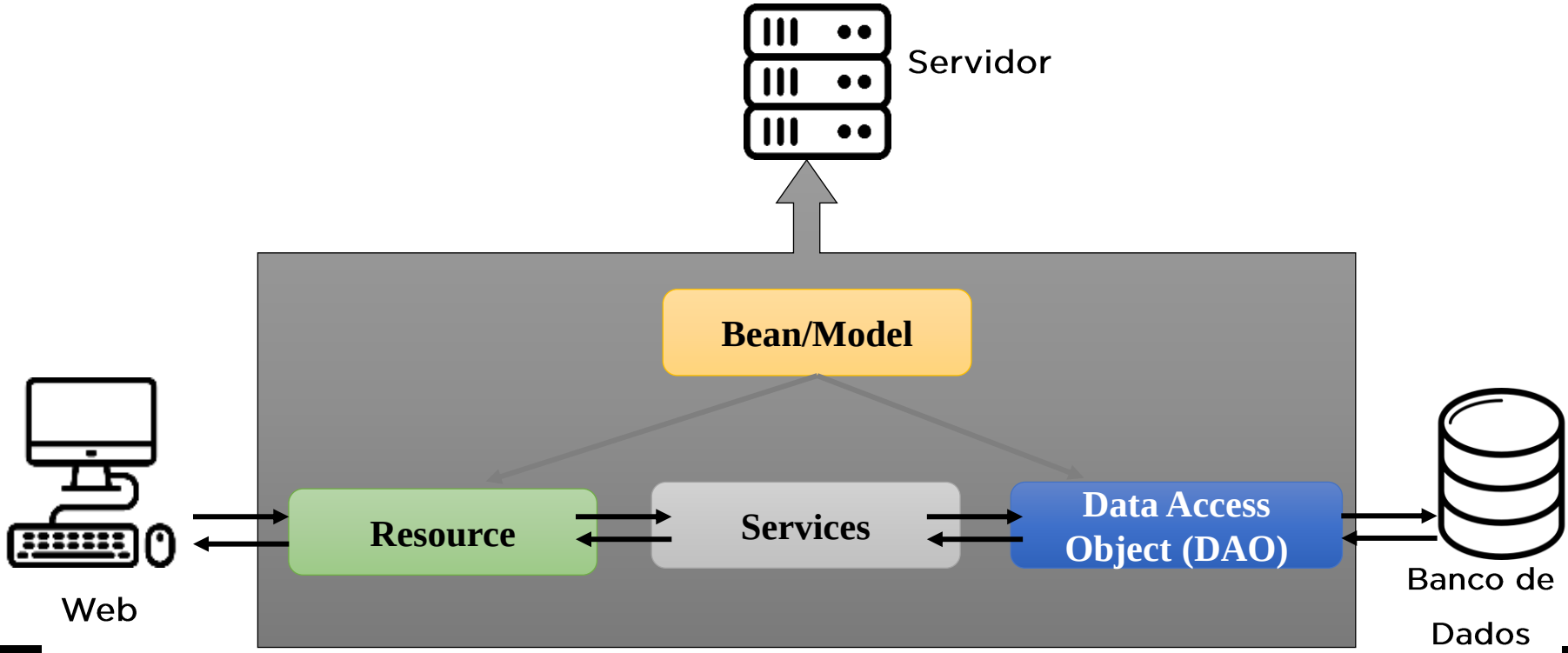
@Path: É usada para definir a rota URI para o recurso.

@GET, @POST, @PUT, @DELETE indicam o tipo de operação HTTP

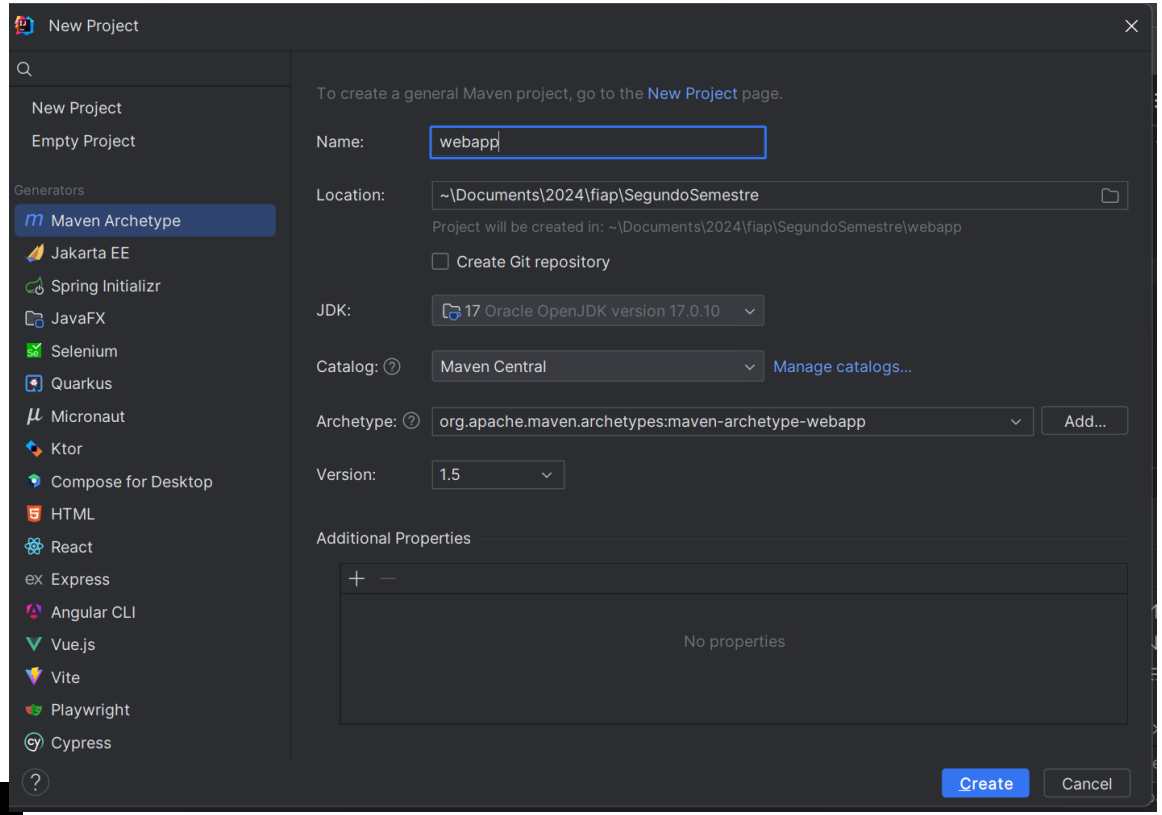
04

Integração de aplicações

DOMAIN DRIVEN DESIGN



CRIANDO O PROJETO



Para criarmos uma aplicação web, abra o IntelliJ e selecione **New Project.**

1º Passo: Escolha a opção **Maven Archetype**

2º Passo: Defina **o nome do projeto.**

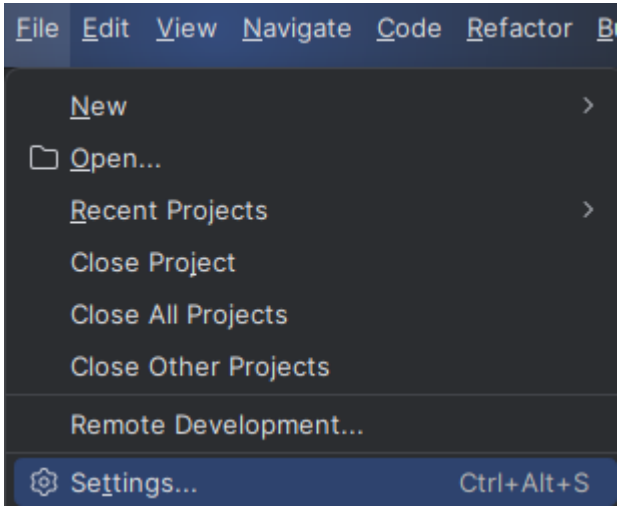
3º Passo: Na opção **Catalog** escolha **Maven Central.**

4º Passo: Na opção **Archetype** escolha **org.apache.maven.archetype:maven-archetype-webapp.**

Clique em Create.

ADICIONANDO O SERVIDOR TOMCAT

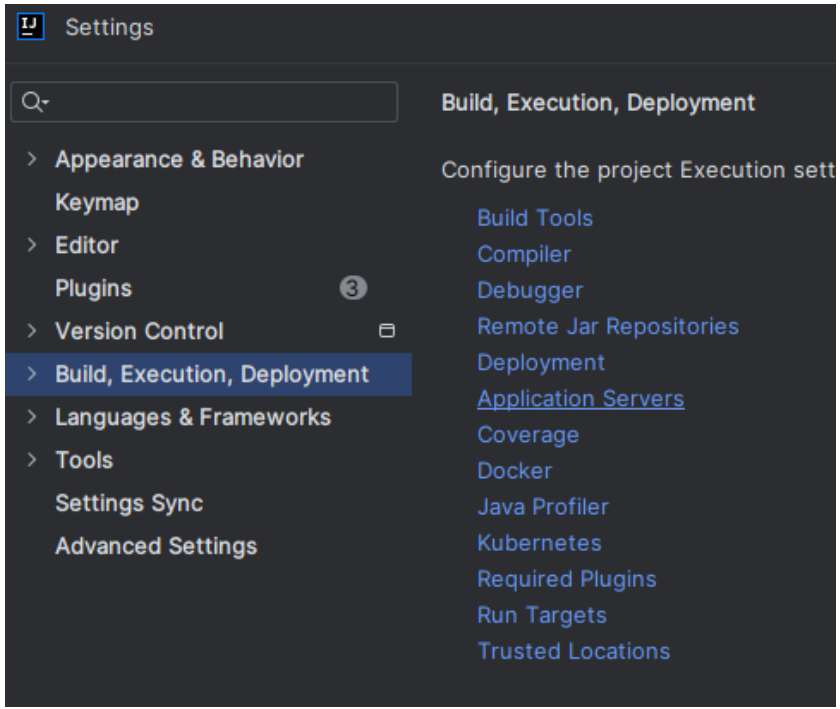
Caso esteja utilizando a versão **Ultimate** do IntelliJ



Clique em File -> Settings

ADICIONANDO O SERVIDOR TOMCAT

Caso esteja utilizando a versão **Ultimate** do IntelliJ



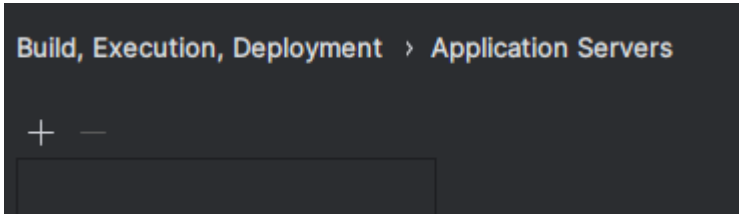
Clique em **Build, Execution, Deployment -> Application Servers**

Faça o download do **Apache Tomcat 10** no site:

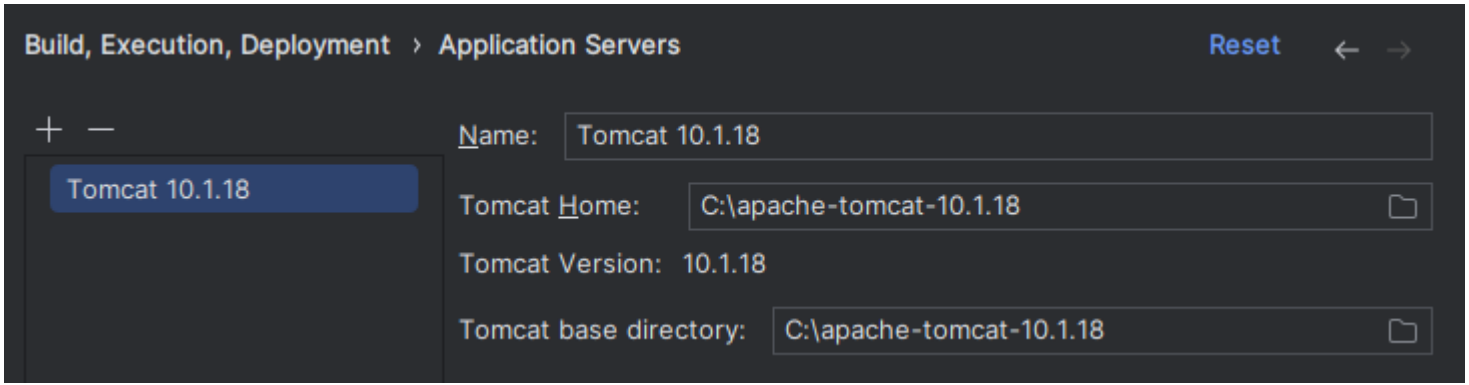
https://tomcat.apache.org/download-10.cgi

Descompacte o arquivo .zip em algum diretório

CONFIGURANDO O TOMCAT



Clique no **+** para adicionar o servidor .
Selecione Tomcat Server

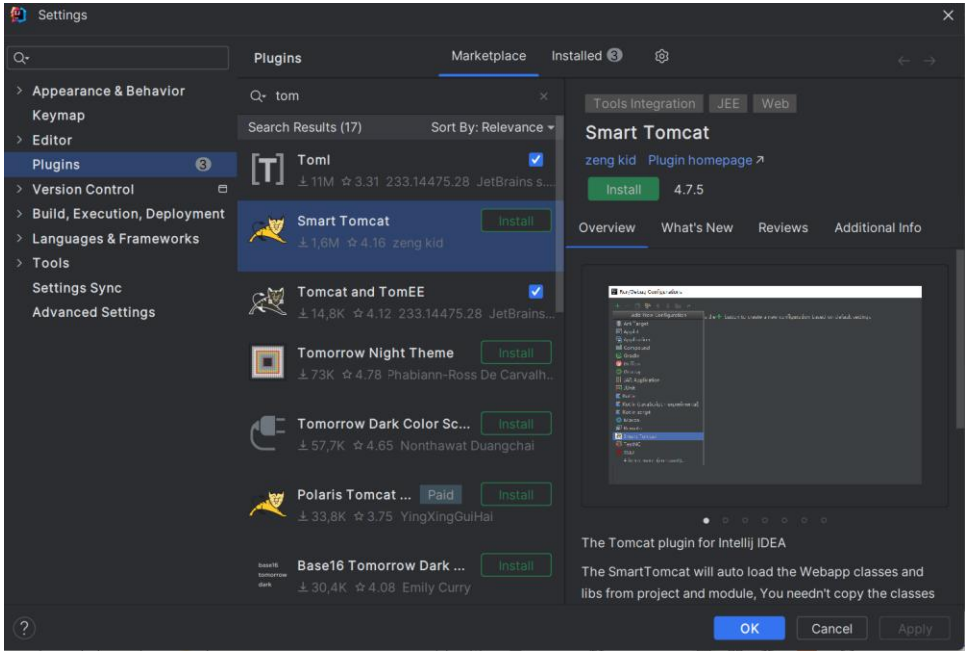


Na opção **Tomcat Home** -> clique no ícone de pasta e aponte para o diretório que baixou o tomcat.

ADICIONANDO O SERVIDOR TOMCAT

IntelliJ Community

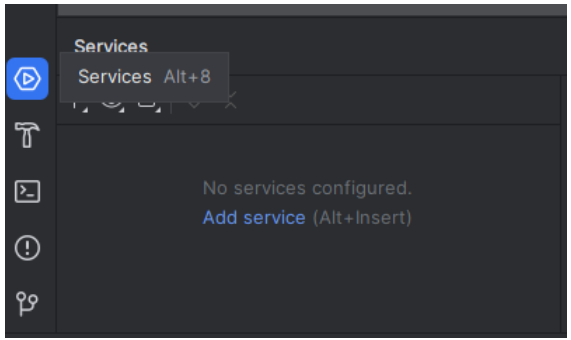
- Clique em **Settings** e selecione a opção **Plugins.**
- Instale o **Smart Tomcat**

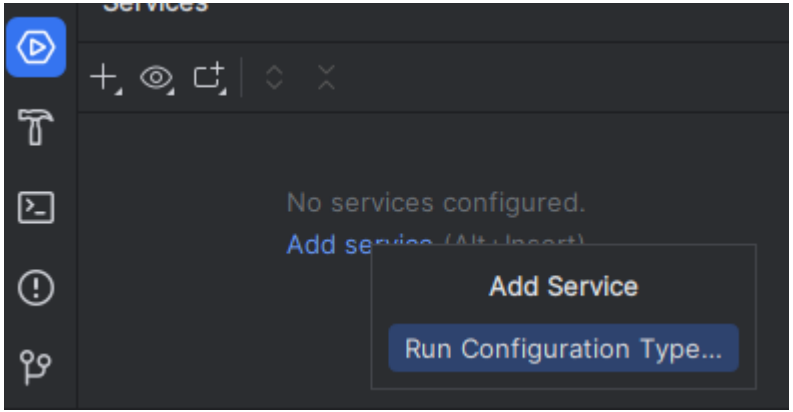


ADICIONANDO O SERVIDOR TOMCAT

IntelliJ Community

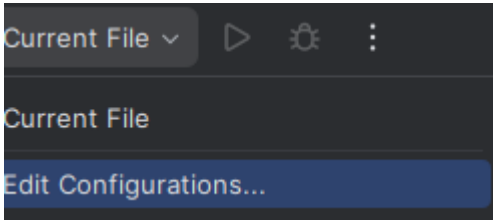
Clique no ícone Services e em Add Service

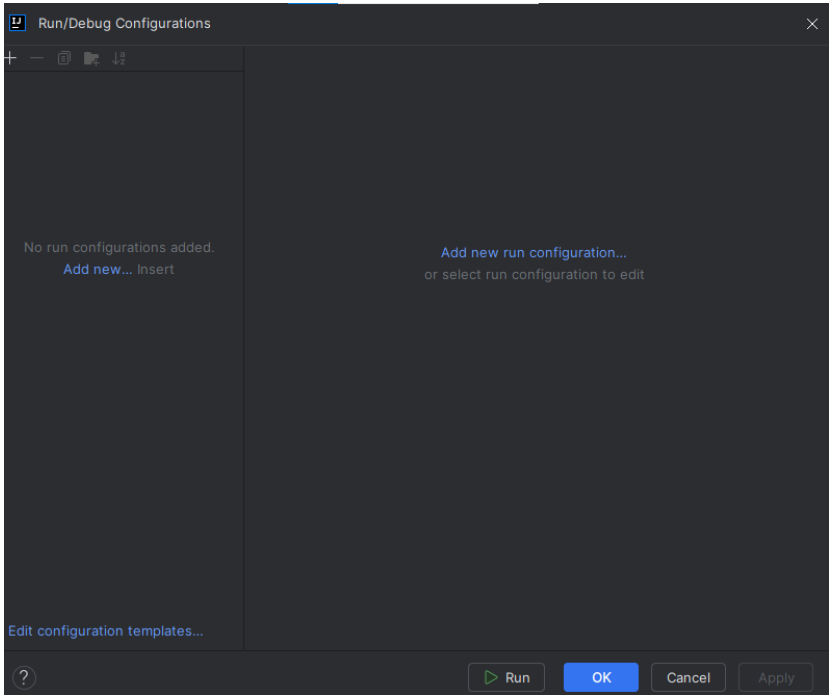




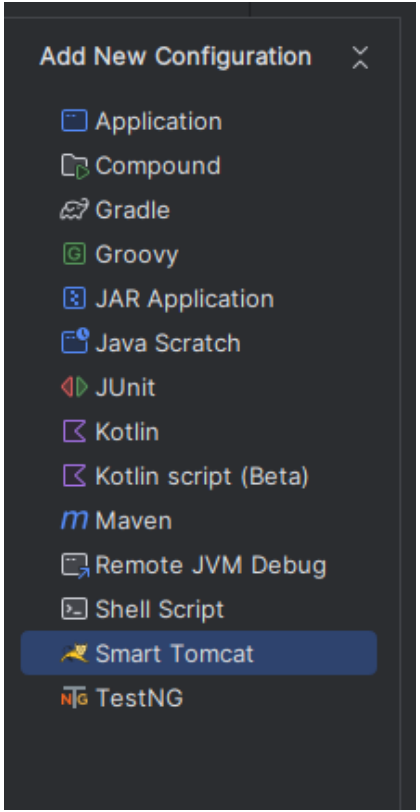
ADICIONANDO O SERVIDOR TOMCAT

IntelliJ Community

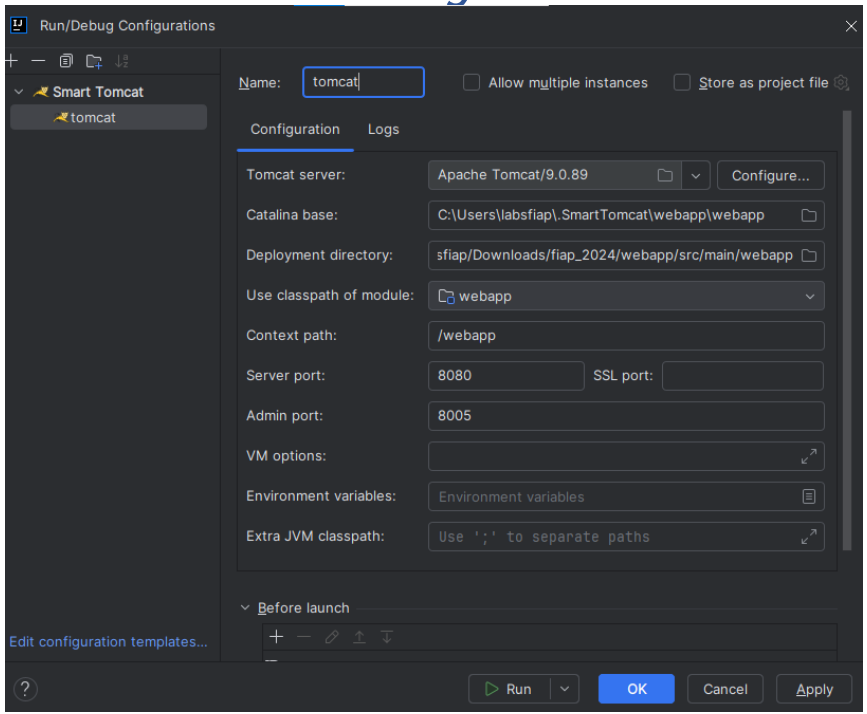




ADICIONANDO O SERVIDOR TOMCAT



IntelliJ Community



ADICIONE AS DEPENDÊNCIAS – **POM.XML**

```
<dependencies>
	<dependency>
		<groupId>org.glassfish.jersey.core</groupId>
		<artifactId>jersey-server</artifactId>
		<version>3.1.1</version>
	</dependency>
	<dependency>
		<groupId>org.glassfish.jersey.containers</groupId>
		<artifactId>jersey-container-servlet</artifactId>
		<version>3.1.1</version>
	</dependency>
	<dependency>
		<groupId>org.glassfish.jersey.media</groupId>
		<artifactId>jersey-media-json-jackson</artifactId>
		<version>3.1.1</version>
	</dependency>
	<dependency>
		<groupId>org.glassfish.jersey.inject</groupId>
		<artifactId>jersey-hk2</artifactId>
		<version>3.1.1</version>
	</dependency>
</dependencies>
```

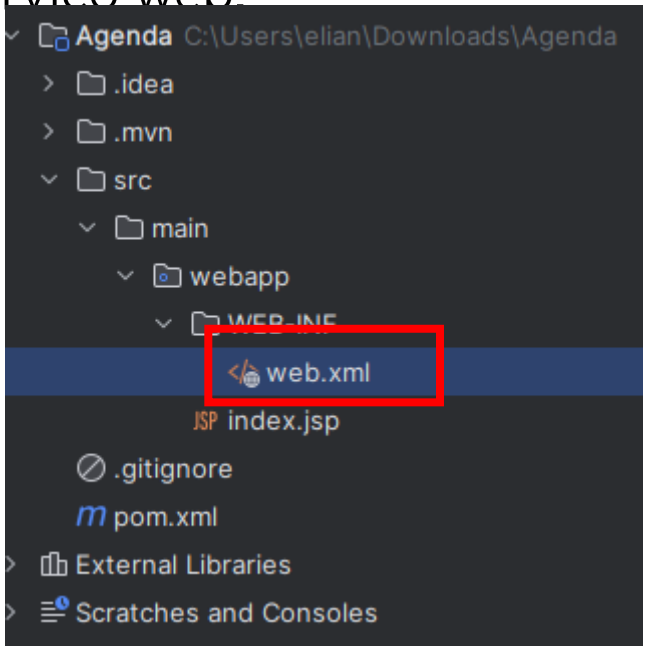
ADICIONE AS DEPENDÊNCIAS – **POM.XML**

- Além das dependências do **JAX-RS**, é preciso adicionar o **driver para acessar o banco de dados:**

```
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc11</artifactId>
    <version>23.2.0.0</version>
</dependency>
```

CONFIGURAÇÃO – **WEB.XML**

- No arquivo **web.xml**, devemos configurar o projeto para atender as requisições do serviço web:



CONFIGURAÇÃO – **WEB.XML**

```
<servlet>
     <servlet-name>jersey-servlet</servlet-name>
     <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
     <init-param>
          <param-name>jersey.config.server.provider.packages</param-name>
          <param-value>br.com.fiap.resource</param-value>
     </init-param>
     <init-param>
          <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
          <param-value>true</param-value>
     </init-param>
     <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
     <servlet-name>jersey-servlet</servlet-name>
     <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

Pacote onde estão as classes do web services

Parte da URL para acessar o web service

JAX-RS – **ANNOTATIONS**

- Principais anotações:

Anotação	Descrição
@Path	Define o caminho para o recurso (URI).
@POST	Responde por requisições POST.
@GET	Responde por requisições GET.
@PUT	Responde por requisições PUT.
@DELETE	Responde por requisições DELETE.
@Produces	Define o tipo de informação que o recurso retorna.
@Consumes	Define o tipo de informação que o recurso recebe.
@PathParam	Injeta um parâmetro da URL no parâmetro do método.

JSON e JAVA

- Vamos trabalhar com a biblioteca **Jackson** para converter objetos **Java** em representações **Json** e vice-versa.
- A **dependência** já foi adicionada no projeto, dessa forma a biblioteca irá realizar a **conversão**

```
public class Produto {

        private int codigo;

        private String nome;

        private double preco;

        private int quantidade;


        //construtores, gets e sets;

}
```

Crie a classe para armazenar as informações do produto.

LISTAR PRODUTOS - **ProdutoResource**

{}

```java
@Path("/produtos")
public class ProdutoResource {

    private ProdutoService service = new ProdutoService();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<ProdutoRequestDto> listar(){
        return service.listar();
    }
```

Tipo do retorno (JSON)

Retorna a lista de produtos para ser convertido em um JSON array.

LISTAR PRODUTOS - **ProdutoService**

{}

```
public List<ProdutoRequestDto> listar(){
   List<Produto> produtos = produtoDao.listar();
   return produtos.stream()
         .map(produto -> {
            ProdutoRequestDto produtoDto = new ProdutoRequestDto();
            produtoDto.setCodigo(produto.getCodigo());
            produtoDto.setNome(produto.getNome());
            produtoDto.setPreco(produto.getPreco());
            produtoDto.setQuantidade(produto.getQuantidade());
            return produtoDto;
         })
         .collect(Collectors.toList());
}
```

TESTE – MÉTODO **GET**

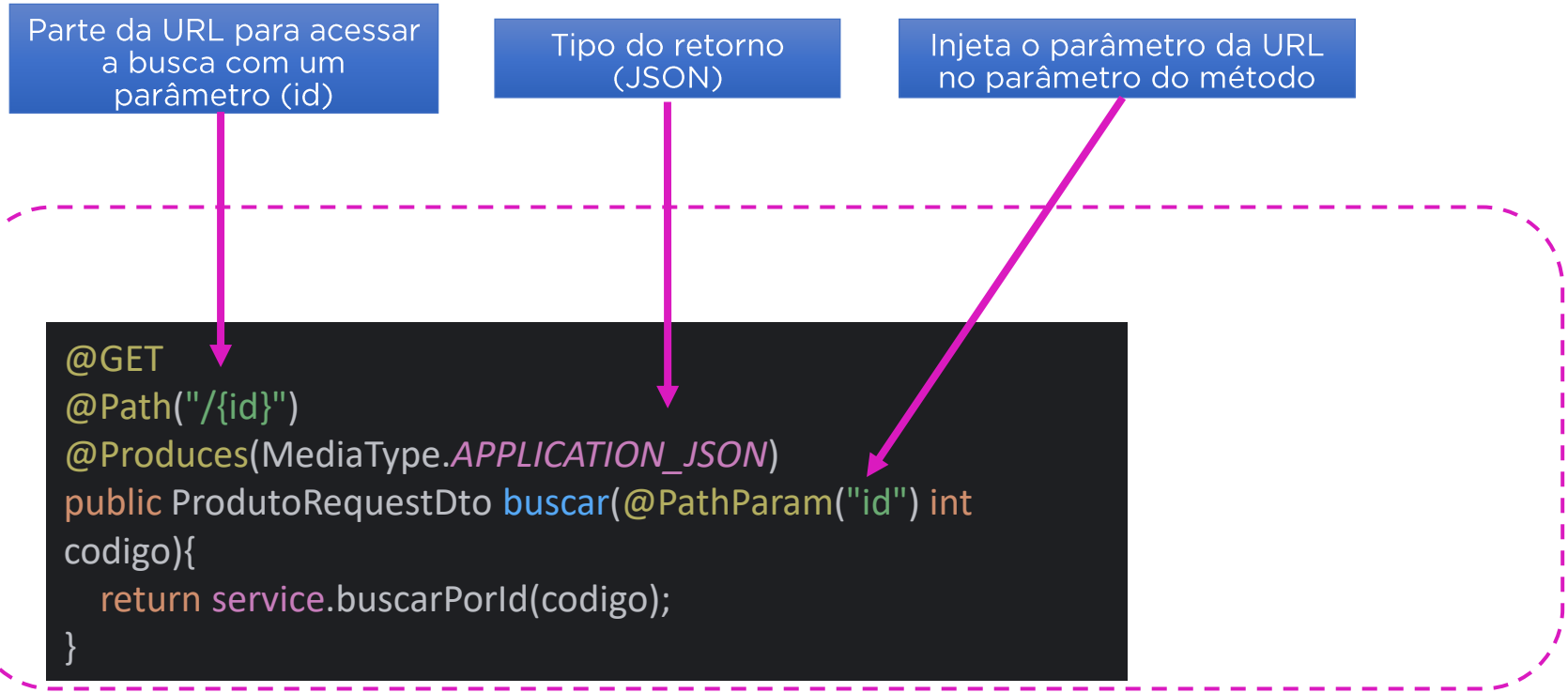
- O **método HTTP padrão** para acessar um endereço através do browser é o GET;
- Execute o Servidor e faça uma chamada ao serviço através de sua **URL** no navegador:

http://localhost:8080/webapp_war/api/produtos

Protocolo, Host, Porta e Contexto | Definido no **web.xml** | Definido no @Path

MÉTODO **GET – BUSCAR POR CÓDIGO**

- Adicione um serviço para **recuperar um produto** pelo seu código.



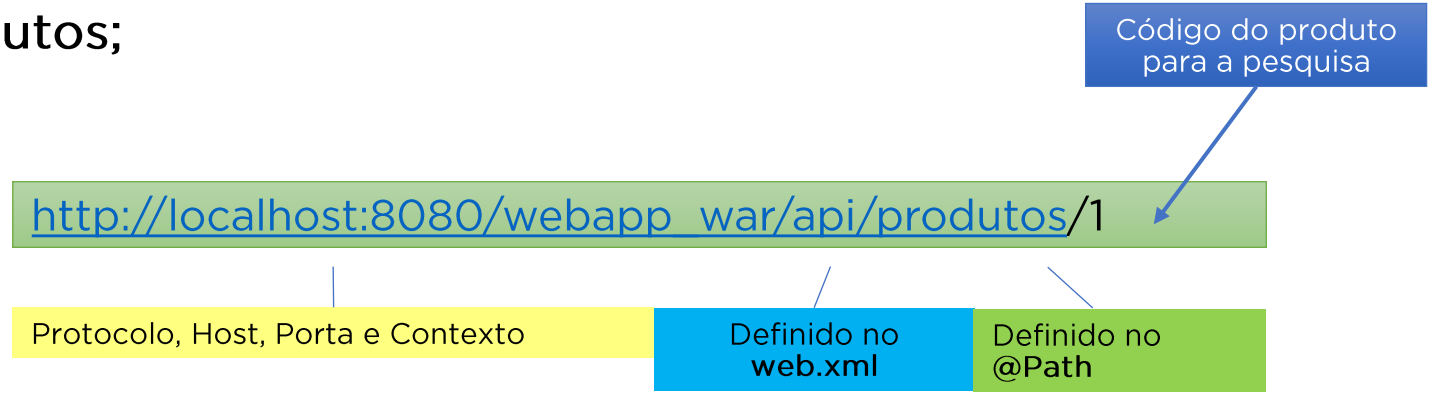
BUSCAR POR CÓDIGO - **ProdutoService**

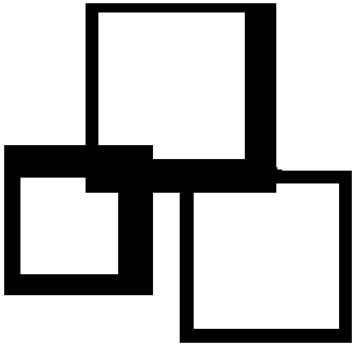
{ }

```
//Retornar um único produto -> buscarPorId
public ProdutoRequestDto buscarPorId(int codigo){

    Produto produto = produtoDao.buscarPorId(codigo);
    ProdutoRequestDto produtoRequestDto = new ProdutoRequestDto();
   return produtoRequestDto.convertToDto(produto);

}
```

TESTE – **GET** BUSCAR POR CÓDIGO

- Para realizar a **pesquisa por código**, basta adicionar o código do produto na **URL**;
- Dessa forma, se você informar o código, o web service **busca um produto específico**, caso não informe, é retornado **todos os produtos;**





OBRIGADO

To be continued...