

# StartDB – 2024/2

## Introdução ao Spring Boot

---

Módulo 2 – Aula 1

Adaptado: Prof. Julio Machado

Profa. Dr<sup>a</sup>. Eduarda Rodrigues Monteiro

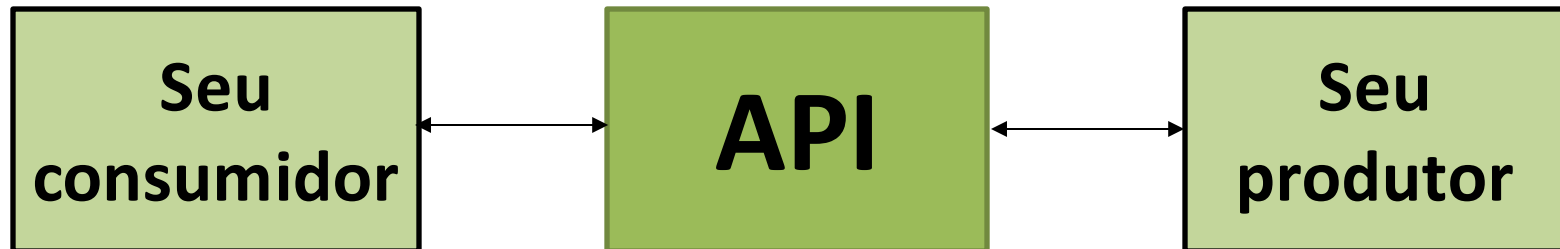
E-mail: [eduarda.monteiro@pucrs.br](mailto:eduarda.monteiro@pucrs.br)

Porto Alegre, Novembro de 2024.

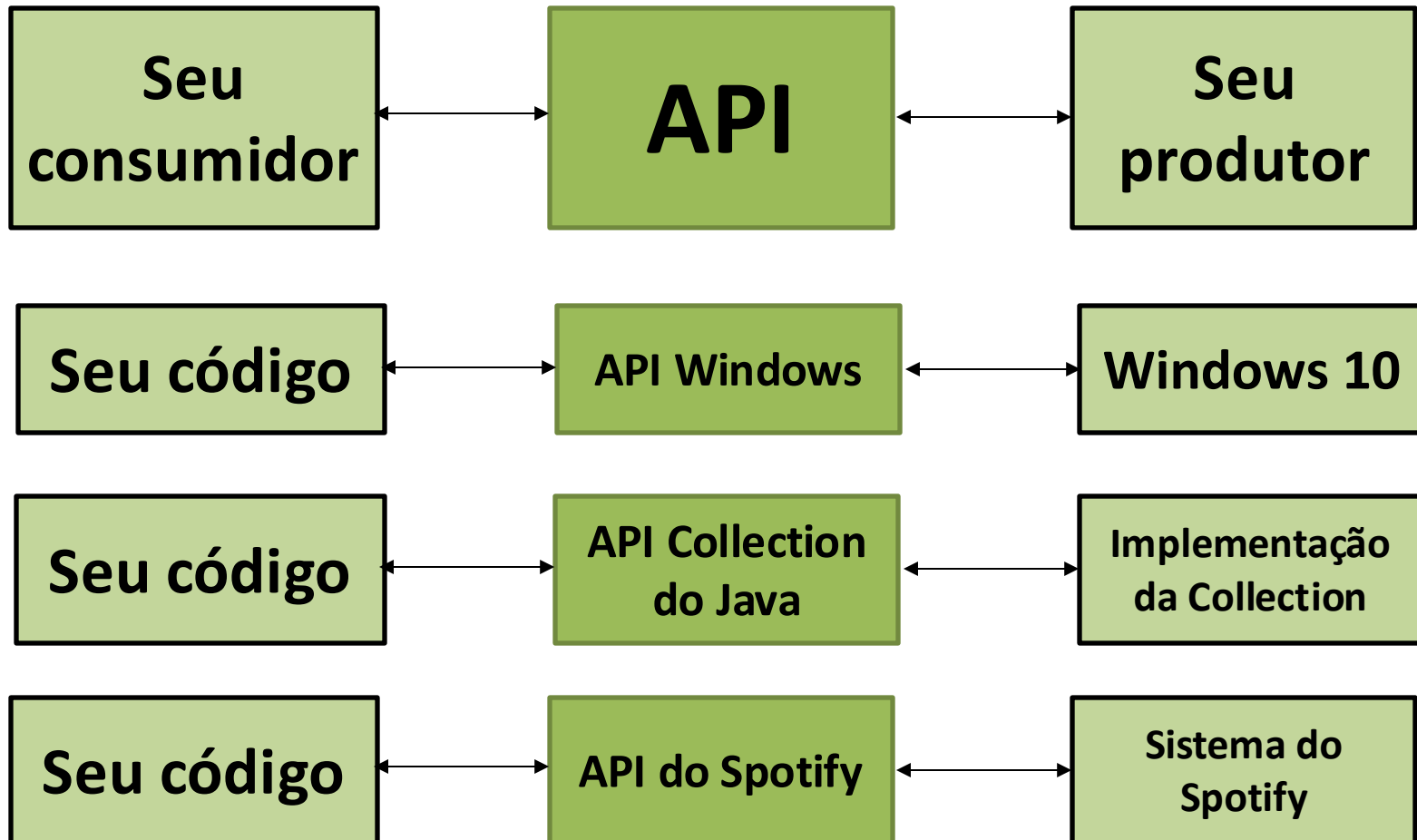
# O que é uma API?

---

- *Application Programing Interface*
- É um software que possui um conjunto de funções implementadas as quais intermediam o acesso às funcionalidades entre sistemas

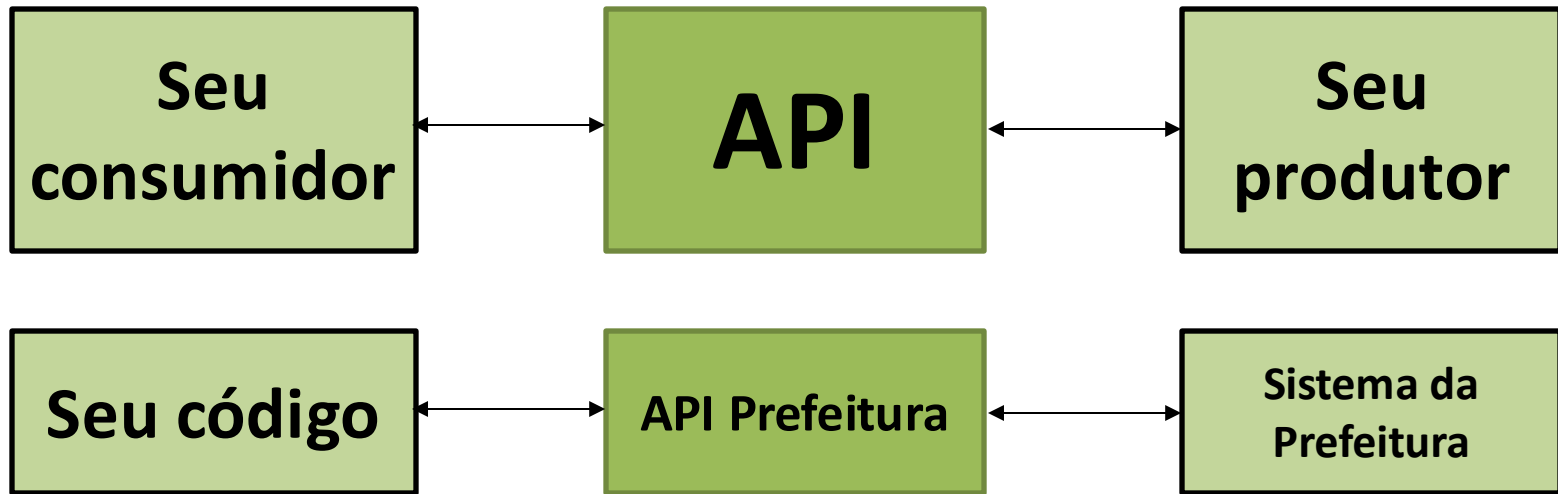


# Exemplos



# Exemplos

---



# Web Services vs API

---

- Web Services
  - É uma API que fornece a interface de comunicação via Internet
  - Todo Web Service é uma API
  - Mas nem toda API é uma Web Service
    - Uma classe pode ser considerada uma API e ela não é web
  - Também conhecido como Web APIs

# Web Services vs API

---

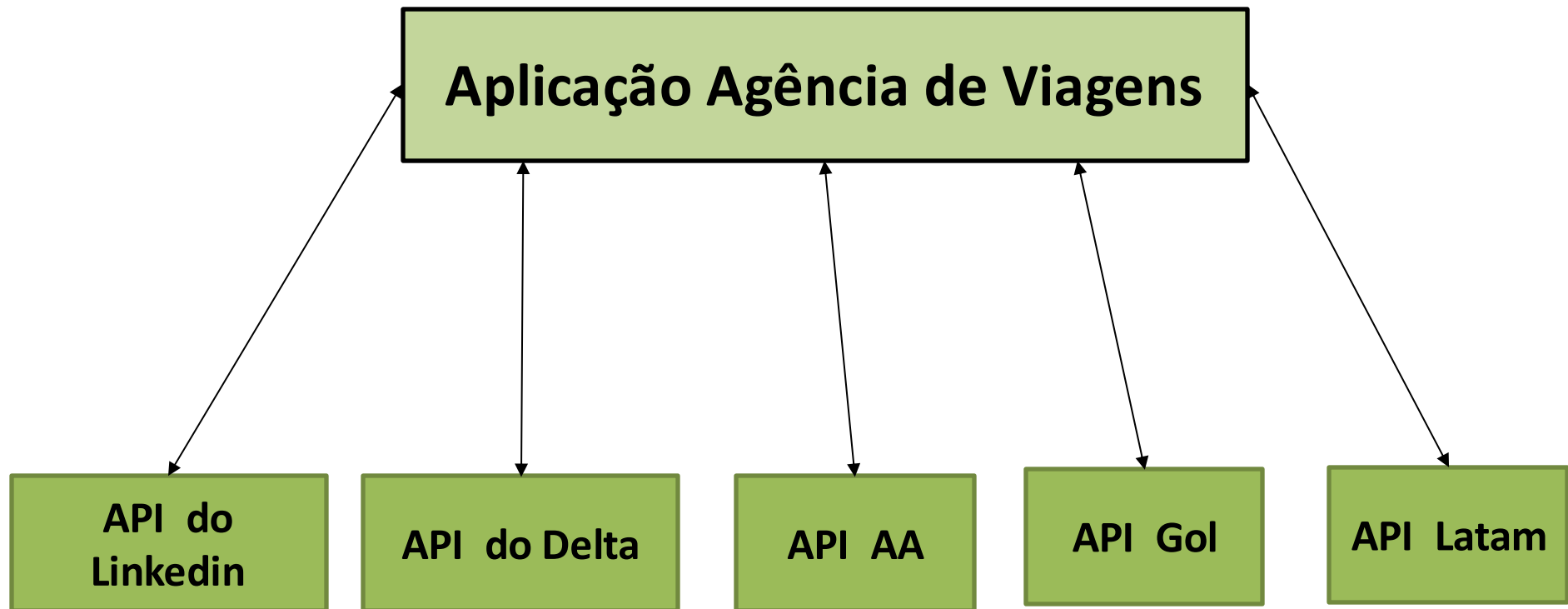
- Consumindo Web Services de um provedor



# Web Services vs API

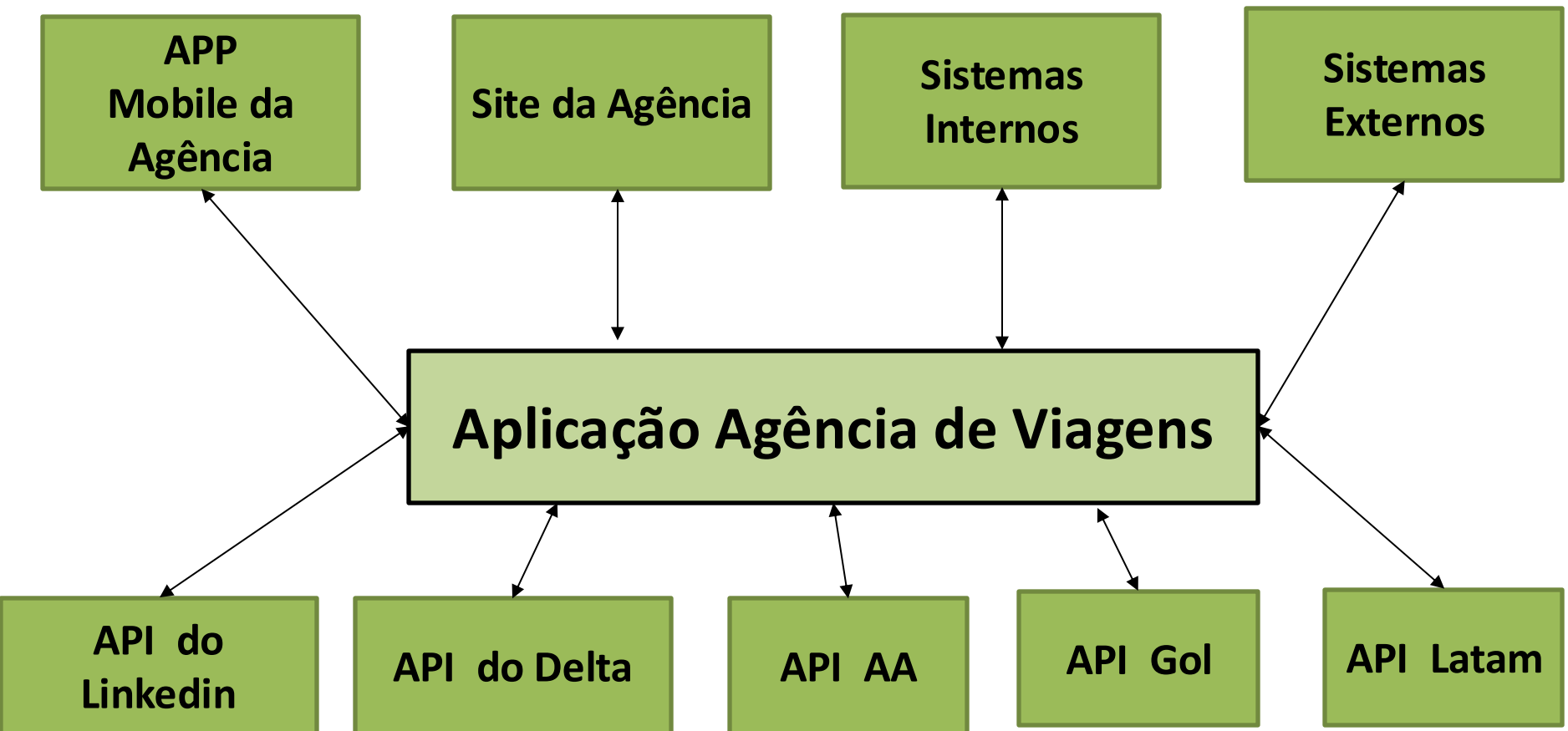
---

- Consumindo Web Services de vários provedores



# Web Services vs API

- APIs são consumidas por suas User Interfaces e outros sistemas.





# Exemplos de APIs

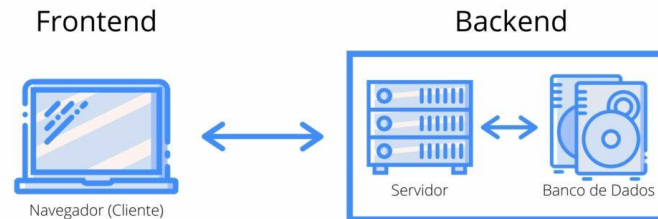
---

- Spotify
- Ifood
- Portal da Transparência
- Mercado Livre

**Poderiam ser úteis para?**

# Rest

- *Representational State Transfer*
- Estilo Arquitetural para desenvolver Web Services (Web API)
- Principal recurso: uso de protocolo que já existe → HTTP



- Para uma aplicação ser considerada Rest a mesma deve seguir algumas regras

# Rest

---

- Representational State Transfer
- Estilo Arquitetural para desenvolver Web Services (Web API)
- Principal recurso: uso de protocolo que já

**Será que desenvolver Rest APIs  
é um bom começo?**

mesma deve seguir algumas regras

**Pequenas e grandes empresas desenvolvem  
e consomem APIs o tempo todo....**

# O que é Spring?

---

- É uma tecnologia de backend
- Não é apenas um framework : é um conjunto de projetos que resolvem vários problemas do cotidiano de um programador
- Ajuda a criar aplicações Java com simplicidade e com flexibilidade
- Conjunto de projetos: Ecossistema Spring
- Ideia: que o Spring nos ajude a criar regras de negócio, de forma que a gente não perca tempo desenvolvendo código da infraestrutura da aplicação

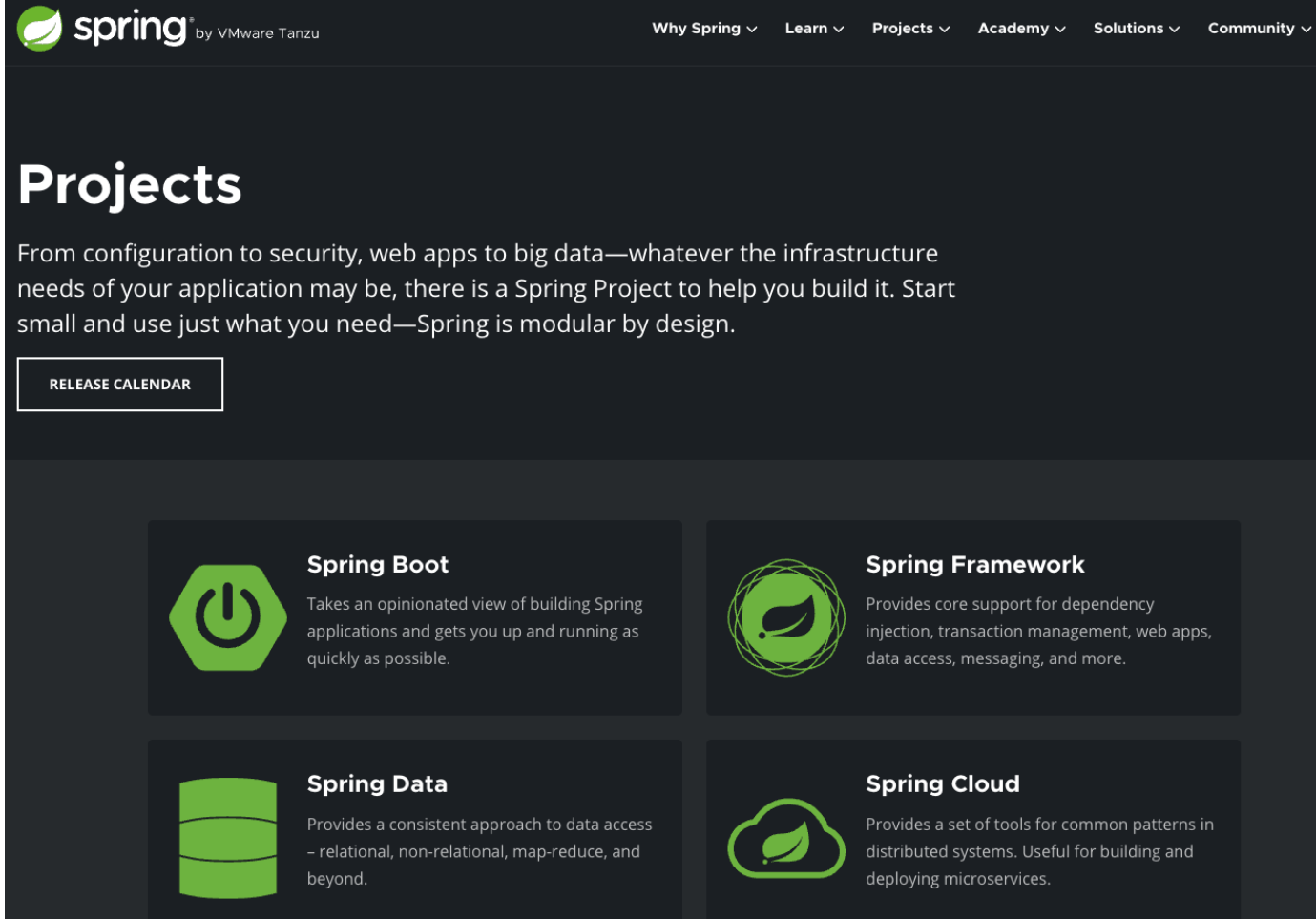
# Por que Spring?

---

- Canivete suíço para desenvolvedores Java
  - Resolve vários e diferentes problemas
- Simplifica o desenvolvimento
- Maturidade
  - Utilizado pelas grandes empresas do mundo
- Modular
  - Adiciona somente módulo que necessita
  - Heterogeneidade: uso com outras tecnologias
- Evolução Constante
- Open Source
- Popularidade

# Ecossistema Spring

- Acessar [spring.io](https://spring.io)



The screenshot shows the Spring.io website with a dark theme. At the top is the Spring logo (a green leaf) and the text "spring by VMware Tanzu". To the right of the logo are navigation links: "Why Spring", "Learn", "Projects", "Academy", "Solutions", and "Community", each followed by a downward arrow. Below the navigation bar is a large section titled "Projects" in white. Under this title is a paragraph: "From configuration to security, web apps to big data—whatever the infrastructure needs of your application may be, there is a Spring Project to help you build it. Start small and use just what you need—Spring is modular by design." Below the paragraph is a button labeled "RELEASE CALENDAR". At the bottom of the screenshot are four project cards, each with a green icon and a description:

- Spring Boot**: Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible. (Icon: a green power button symbol inside a hexagon)
- Spring Framework**: Provides core support for dependency injection, transaction management, web apps, data access, messaging, and more. (Icon: a green leaf inside a circular web-like structure)
- Spring Data**: Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond. (Icon: a green database cylinder)
- Spring Cloud**: Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices. (Icon: a green leaf inside a cloud shape)

# Spring Boot

---

- Um dos módulos do projeto Spring
- Auxilia em configurações complexas
- Módulo responsável pela configuração de projeto de forma mais simplificada
- Facilita configuração no uso de outros módulos
- Usa convenções para realizar as configurações de forma automática

# Spring Boot

---

- Exemplo:
  - Toda transação com Banco de Dados tem que ser aberta e fechada
  - Quando a persistência do projeto for configurada essa premissa já estará configurada automaticamente
- Incorpora um Servelet Container que se chama TOMCAT
- O JAR da aplicação já vem com o Tomcat, facilitando o deploy
- Spring Boot não é um gerador de código, apenas auxilia na configuração do projeto



# Abre Parênteses

---

- Tomcat
  - Apache
  - Servlet Container (servidor de aplicações)
  - Servidor Jakarta
- Jakarta
  - Conjunto de Especificações
  - Java EE (Java Plataform, Enterprise Edition)

# Spring Boot

---

- Síntese:
  - *Convention over Configuration*
  - Programador foca mais nas regras de negócios
  - Quando cria o projeto da aplicação precisa apontar o que é necessário para seu projeto
  - Auto configuração
    - Maven/POM.xml
  - Spring boot também facilita o uso do POM através dos Starters

# Spring Boot

---

- O que vamos precisar?!

# Spring Boot

---

- Pré-Requisitos

- Existem diversas formas de trabalhar com Spring Boot.
  - Uma delas exige os seguintes softwares previamente instalados:
    - Java, Vscode e Maven
- Uso de uma ferramenta capaz de simplificar o envio de requisições HTTP para o nosso “backend” sem a necessidade de termos de construir um módulo “frontend”.
  - Em nossos exemplos usaremos a ferramenta “Postman” ([Postman API Platform](#)), porém, existem diversas soluções semelhantes a disposição, tal como [HTTPie](#)

# Criando o Primeiro Projeto

---

- Para criar um projeto Spring Boot usar (alternativas):
  - O Spring Initializer: [Spring Initializr](#)
  - Plugin do VSCode:  
<https://marketplace.visualstudio.com/items?itemName=vmware.vscode-boot-dev-pack>
- Nos dois casos é necessário indicar uma série de informações tais como:
  - Tipo de projeto: “Maven”
  - Versão do Spring-Boot: 3.3.3
  - “Group Id”: seunome
  - Nome do artefato: ex1-biblioteca
  - Packing: jar
  - Versão do Java: 21
  - Dependências: (inicialmente usaremos apenas estas duas)
    - Spring-Boot Dev Tools
    - Spring WEB
- A ferramenta Spring Initializer gera um zip com todo conteúdo do projeto
- O plugin do VSCode já gera a estrutura do projeto no local indicado

# Criando o Primeiro Projeto

## Project

- ☐ Gradle - Groovy  
☐ Gradle - Kotlin ☒ Maven

## Language

- ☒ Java ☐ Kotlin  
☐ Groovy

## Spring Boot

- ☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M2) ☐ 3.3.4 (SNAPSHOT)  
☒ 3.3.3 ☐ 3.2.10 (SNAPSHOT) ☐ 3.2.9

## Project Metadata

Group julioapm

Artifact ex1-biblioeca

Name ex1-biblioeca

Description Demo project for Spring Boot

Package name julioapm.ex1-biblioeca

Packaging ☒ Jar ☐ War

Java ☐ 22 ☒ 21 ☐ 17

## Dependencies

ADD DEPENDENCIES... CTRL + B

## Spring Boot DevTools

DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

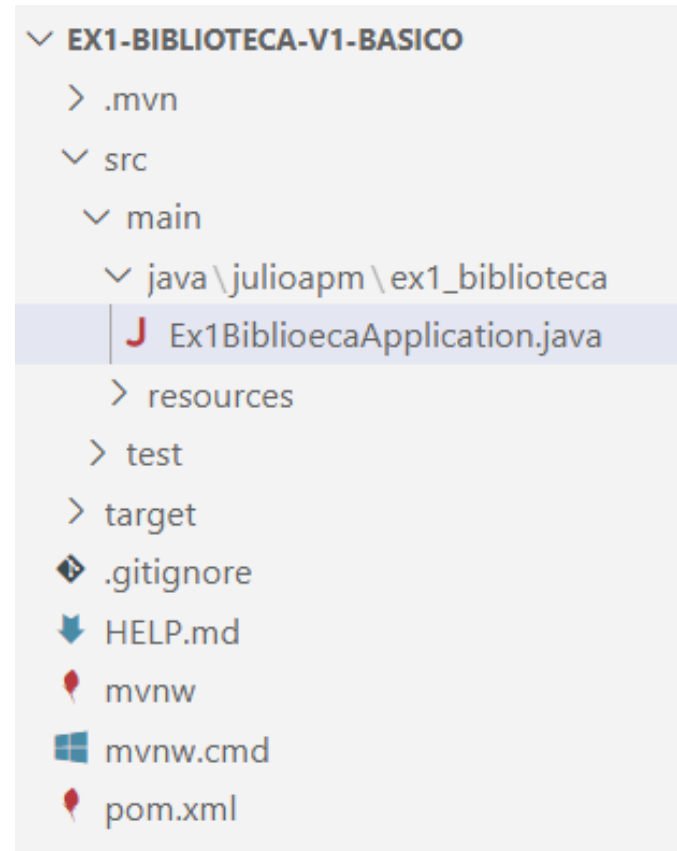
## Spring Web

WEB

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

# Estrutura do Projeto

- O projeto será criado em uma pasta com o nome do projeto: **ex1-biblioteca**
- O programa principal (Ex1BiblioecaApplication.java) será criado na pasta **src**.
- O arquivo “pom.xml” contém toda a descrição do projeto.



# O Arquivo Ex1BibliotecaApplication.java

```
package eduarda.ex1_biblioteca;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication
;

@SpringBootApplication

public class Ex1BiblioecaApplication {

    public static void main(String[] args) {
        SpringApplication.run(Ex1BiblioecaApplication.class, args);
    }
}
```

- O programa principal é responsável por toda as inicializações automáticas do Spring Boot.
- A única anotação necessária, por enquanto, é “@SpringBootApplication” que indica o tipo de aplicação.
- Esta aplicação coloca no ar uma instância do Tomcat, porém, não temos ainda nenhum “endpoint” configurado para atender requisições HTTP.



# Acrescentando um “Controller”

```
package eduarda.ex1_biblioteca;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Controller {
    @GetMapping
    public String mensagemDeBemVindo() {
        return "Bem vindo a biblioteca central!";
    }
}
```

- O arquivo “Controller.java” implementa um “controller”.
- A identificação da classe que implementa um “controller” é feita pela anotação “@RestController”
- Cada função anotada com “@GetMapping” indica um endpoint
  - O roteamento correspondente é anotado no parâmetro.
- No exemplo ao lado não temos nenhuma rota anotada, logo este será o endpoint padrão quando nada for indicado

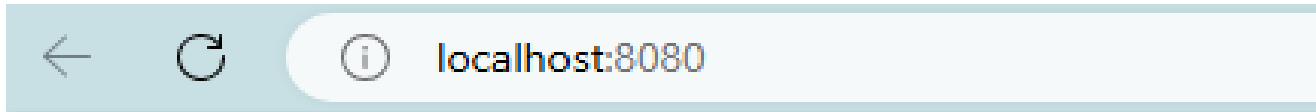
# Executando o projeto

---

- Para executar o projeto localize o console na pasta raiz do projeto (aquela que tem o arquivo “pom”) e digite: `mvn spring-boot:run`
- O projeto irá compilar e executar. A execução começa colocando no ar uma instância do Tomcat e depois fica escutando a porta 8080 aguardando eventuais requisições.
- Se estiver executando o programa na máquina local, abra o navegador e acesse a URL: <http://localhost:8080>
- Se estiver executando no Codespaces, selecione a aba “ports” do terminal e clique no link exibido.

# Visualização do Projeto

---



Bem vindo a biblioteca central!

Utilizar diretamente o navegador para testar um endpoint de um web service HTTP não é uma boa prática!

O navegador “esconde” toda a estrutura do protocolo HTTP.

# Interrompendo a execução

---

- Para parar a execução digite “Ctrl+C” no console do terminal.
- Para garantir que a aplicação libere a porta execute: `mvn spring-boot:stop`

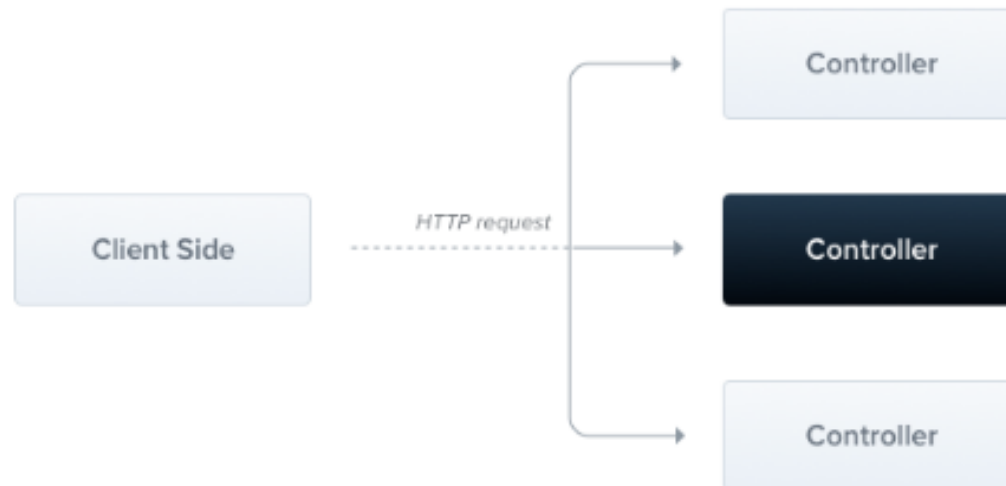
# Vamos entender o que ocorreu....

---

- Uma aplicação “backend” não tem “interface com o usuário”
- Ao invés disso ela implementa uma série de “endpoints”
- Cada “endpoint” é acionado em resposta a um tipo diferente de requisição (no caso requisições HTTP)
- A(s) classe(s) que implementam os “endpoints” usualmente são chamadas de “controllers”
- Os “controllers” são a forma de comunicação das aplicações “backend” com o mundo externo

# Entendendo os “Controllers”

- Uma aplicação pode ter vários “controllers” de maneira a lidar com subconjuntos de requisições
- Nossas primeiras aplicações terão apenas 1 “controller”



# Requisição sem Rotas

---

- Até o momento nossa aplicação só é capaz de atender uma única requisição HTTP: <http://localhost:8080>
- Esta é uma requisição GET simples
  - Não tem “subcaminhos”, (o path é apenas “/”)
  - Não tem “parâmetros de rota”
- Para entendermos como fazer para atender requisições mais elaboradas vamos revisar o protocolo HTTP

# HTTP: Introdução

---

- Protocolo *Hypertext Transfer Protocol*
- Viabiliza a comunicação entre clientes e servidores de maneira simples;
- Sem manutenção de estado;
- Utiliza um formato de texto
- Um modelo de requisição/resposta.
- É fundamental para o funcionamento da web, facilitando a troca de dados e o acesso a conteúdos de maneira eficiente.



# HTTP: Introdução

---

- Características
  - Protocolo de nível de aplicação
  - Protocolo textual
  - Protocolo baseado em mensagens de requisição/resposta no modelo cliente/servidor
  - Protocolo sem manutenção de estado

# HTTP: Introdução

---

- Características
  - Protocolo de nível de aplicação
    - No modelo OSI (*Open Systems Interconnection*) é um protocolo da camada de aplicação.
    - Camada mais próxima do usuário
    - Protocolos de aplicação são usados para comunicação direta entre aplicativos, facilitando o envio e recebimento de dados.

# HTTP: Introdução

---

- Características
  - Protocolo de nível de aplicação
  - **Protocolo textual**
    - Podemos enviar mensagens através do HTTP:
    - Essas mensagens são enviadas e recebidas em formato de texto legível
    - As requisições HTTP: GET, POST, PUT, etc.
    - As respostas do servidor terão um código (como os códigos de status '200 OK', '404 Not Found', etc.)

# HTTP: Introdução

---

- Características
  - Protocolo de nível de aplicação
  - Protocolo textual
  - **Protocolo baseado em mensagens de requisição/resposta no modelo cliente/servidor**
    - HTTP segue um modelo de requisição/resposta, onde o cliente (como um navegador) envia uma requisição ao servidor, e o servidor responde com os dados solicitados.
    - Esse modelo é assíncrono, pois o cliente faz a requisição e aguarda a resposta do servidor.
      - Exemplos:
        - » O cliente solicita uma página usando GET.
        - » O servidor responde com o conteúdo da página ou com um código de erro.

# HTTP: Introdução

---

- Características

- Protocolo de nível de aplicação
- Protocolo textual
- Protocolo baseado em mensagens de requisição/resposta no modelo cliente/servidor

- **Protocolo sem manutenção de estado**

- HTTP é um protocolo sem estado (*stateless*), o que significa que ele não mantém informações entre as requisições.
- Cada requisição HTTP é tratada de forma independente. Isso significa que o servidor não "lembra" das requisições anteriores feitas pelo mesmo cliente.
- Para contornar essa limitação, as aplicações usam mecanismos como cookies, sessões e tokens para manter o estado e rastrear usuários entre requisições.

# HTTP: URL

---

- HTTP foi originalmente criado para lidar com hipertextos
- Neste contexto uma URL *Uniform Resource Locator* foi pensada para identificar arquivos em um servidor Web
  - Ex.: `http://java.com/index.html`
- O conjunto de elementos de uma URL, porém, permitiu que seu uso evoluísse com o tempo
- Estrutura de uma URL: “**protocol:**” “**://**” **host**“**:**” **port**”  
[**path**“**?**” **query**”]
  - Ex: `http://java.com/books/index.html?id=1322`

# HTTP: Estrutura de uma Requisição

---

- Uma requisição HTTP consiste de:
  - Uma linha inicial
  - Um ou mais campos de cabeçalho
  - Uma linha em branco
  - Possivelmente um corpo da mensagem
- Uma resposta HTTP consiste de:
  - Uma linha de status com seu código (ver [RFC](#), [Wikipédia](#)) e mensagem associada
  - Um ou mais campos de cabeçalho
  - Uma linha em branco
  - Possivelmente um corpo da mensagem

# HTTP: Métodos

- Alguns métodos (também chamados de verbos):

|        |   |
|--------|---|
| GET*   | Solicita um recurso ao servidor   |
| POST*  | Fornece a entrada para um comando do lado do servidor e devolve o resultado |
| PUT    | Envia um recurso ao servidor  |
| DELETE | Exclui um recurso do servidor   |
| TRACE  | Rastreia a comunicação com o servidor                                       |

\*métodos mais utilizados para fornecer entrada de dados aos programas no lado servidor



# HTTP: GET

---

- GET:
  - Método mais simples
  - Quantidade de dados muito limitada
    - Limite implementado nos navegadores
  - Dados acrescentados à URL após um caractere “?”, no formato “campo=valor”, separados pelo caractere “&”
    - Recebe o nome de *query-string*
  - Ex.:  
**`http://www.biblioteca.com/livros?autor=Ze&ano=2020`**

# HTTP: POST

---

- Método mais robusto
- Quantidade de dados não é limitada como no GET
- Dados (*query-string*) enviados no corpo da requisição do protocolo
- Permite efeitos colaterais na execução no lado do servidor
- **Requisição:**  
POST /index.html HTTP/1.0  
Accept: text/html  
If-modified-since: Sat, 29 Oct 1999 19:43:31 GMT  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 41  
  
Nome=NomePessoa&Idade=20&Curso=Computacao
- **Resposta:**  
HTTP/1.1 200 OK  
Date: Mon, 23 May 2005 22:38:34 GMT  
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)  
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT  
Etag: "3f80f-1b6-3e1cb03b"  
Accept-Ranges: bytes  
Content-Length: 438  
Connection: close  
Content-Type: text/html; charset=UTF-8

# Incluindo sub-caminhos nos “controllers”

- **No exemplo ao lado** indicamos sub-caminhos no decorador “@GetMapping”
- O caminho indicado em “**@GetMapping("/livros")**” é ativado pela URL que segue:  
<http://localhost:8000/livros>
- Altere e teste o projeto conforme ao lado

```
@RestController
public class Controller {
    @GetMapping
    public String mensagemDeBemVindo() {
        return "Bem vindo a biblioteca central!";
    }
    @GetMapping("/livros")
    public String getLivros() {
        return "Lista de livros";
    }
}
```

# Incluindo sub-caminhos nos “controllers”

- Por vezes as aplicações possuem muitos endpoints e é necessário agrupar os mesmos em sub-caminhos.
- Imagine, por exemplo, que a aplicação ao lado gerencia uma universidade, e que a biblioteca é apenas um dos itens.
- Então define-se um sub-caminho “/biblioteca” a partir do qual concatenam-se os demais.
- Usa-se a anotação “@RequestMapping” para indicar um sub-caminho.
- A partir desta alteração teste as URLs:
  - <http://localhost:8080/biblioteca>
  - <http://localhost:8080/biblioteca/livros>

```
@RestController
@RequestMapping("/biblioteca")
public class Controller {
    @GetMapping
    public String
    mensagemDeBemVindo() {
        return "Bem vindo a biblioteca
        central!";
    }

    @GetMapping("/livros")
    public String getLivros() {
        return "Lista de livros";
    }
}
```

# Exercício

---

- Vamos relembrar alguns conceitos e estruturas de programação orientada a objetos criando uma resposta melhor para o caminho “/biblioteca/livros”.
- O objetivo deste caminho é retornar a lista de livros contidos no acervo da biblioteca
- Para tanto acrescente uma lista de livros como propriedade privada da classe “Controller” e inicialize a mesma no método construtor desta classe.
- Não se esqueça de criar uma classe que modele um livro (id, título, autor, ano)
- Na sequência altere o método “getLivros” para que retorne a lista de livros contida no acervo. Não se esqueça de alterar o parâmetro de retorno do método para “List<Livro>”.

# Observações

---

- Note como o Spring Boot automatiza a questão das respostas às requisições HTTP.
- Observe como a lista de objetos, é serializada automaticamente para um formato JSON.

# Praticando...

---

- Acrescente dois novos caminhos na aplicação da biblioteca sendo desenvolvida:
  - /biblioteca/titulos → devolve a lista dos títulos pertencentes ao acervo
  - /biblioteca/autores → devolve a lista dos autores dos livros pertencentes ao acervo (sem repetição)